MASTER IN BUSINESS ENGINEERING PROFESSIONAL FOCUS IN DATA SCIENCE

What are the main steps needed to build and create a mobile application using deep learning with the network stored directly on the device?

Vagenende, Dimitri

Award date:
2021

Awarding institution:
University of Namur

What are the main steps needed to build and create a mobile application using deep learning with the network stored directly on the device ?

Dimitri VAGENENDE

Directeur : Prof. B. FRENAY
Co-directeur : J. FINK

Mémoire présenté
en vue de l'obtention du titre de
Master 120 en ingénieur de gestion, à finalité spécialisée
en data science

ANNEE ACADEMIQUE 2020-2021

Université de Namur, ASBL
Faculté des Sciences économiques, sociales et de gestion – Département des Sciences de gestion

Rempart de la Vierge 8, B-5000 Namur, Belgique, Tel. +32 [0]81 72 49 58/48 41

Abstract

The world is experiencing the emergence of the use of new technologies. This one is strongly supported by the massive diffusion of cell phones, the constant increase of available data and by numerous advances in the field of artificial intelligence. This technological breakthrough has opened up numerous possibilities in the decision-making process. Indeed, this can more easily rely on technology. This thesis focuses on the construction of guidelines for the creation of a mobile application using deep learning with the model stored within the mobile. After theoretical points about machine learning and deep learning, this thesis discusses the steps to create the convolutional neural network model and those needed to create the application. In order to help people, specialists or not, to create their own mobile application and to visualize it further, a finished application ready to be deployed will be proposed throughout this thesis.

Résumé

Le monde actuel connait l'émergence de l'utilisation de nouvelles technologies. Cette dernière est fortement portée par la diffusion massive du téléphone portable, de l'accroissement constant des données à disposition et par de nombreuses avancées dans le domaine de l'intelligence artificielle. Cette percée technologique a pour conséquence d'ouvrir de nombreuses possibilités dans l'aide à la prise de décision. En effet, cette dernière peut plus facilement s'appuyer sur la technologie. Ce mémoire porte sur la construction d'une ligne de conduite pour la création d'une application mobile utilisant du deep learning avec le modèle stocké au sein du mobile. Après un point théorique, ce mémoire aborde les étapes pour créer le modèle du réseau de neurones artificiel et celles nécessaires à la création de l'application. Afin d'aider au mieux les personnes, spécialistes ou non, à créer leur propre application mobile et à visualiser davantage, une application finie et prête au déploiement sera proposée tout au long de ce mémoire.

Acknowledge

First, I would like to thank my supervisor, Mr. Frenay, and my co-supervisor, Mr. Fink, for their help, guidance, and patience. Secondly, I would also like to say thank you to my sister, Caroll Vagenende for the help in writing my thesis. Finally, I would like to thank Ms. Bourguigon for drafting a legal document that was necessary for the successful completion of my thesis.

Contents

Chapter 1

Introduction

1.1 Context

The world is experiencing the emergence of the use of new technologies. This one is strongly supported by the massive diffusion of cell phones, the constant increase of available data and by numerous advances in the field of artificial intelligence. This technological breakthrough has opened up numerous possibilities in the decision-making process. Indeed, this can more easily rely on technology. As a result of this favorable context, it is not uncommon to be helped for a daily task by a new technology.
Despite this massive diffusion of technology in society, the underlying understanding of how they work remains a mystery for many people. Therefore, the objective of this thesis is to create a methodology to easily design a mobile application using artificial intelligence and more precisely, deep learning. Until now, few scientific articles have been written on this subject. Indeed, most of the current articles focus on a new contribution given in this field. Therefore, the operational aspect is not yet strongly developed. The main goal of this thesis is to contribute to the development of all scientific sources in this field in the most popularized way possible. This thesis aims to put in writing practices, concepts and an approach to follow in the development of a mobile application. It is summary of knowledge known in this field in order to help the scientific community to relate all of them. If the writing of the approach to follow is based on existing sources, the essential is the fruit of my research and reflections because there exists few sources treating precisely the subject. To summarize, project tends help anyone who wants to understand and start the creation of a mobile application based on deep learning.

1.2 Research question

The writing of this thesis therefore focuses on creating a guideline for creating applications using deep learning. To do so, the whole machine learning model is stored in a cell phone and not stored on an external server. This choice is made because the management of the machine learning model on the phone itself offers several advantages that are explained in the section 4.2. The question that the thesis tries to answer is:

What are the main steps needed to build and create a mobile application using deep learning with the network stored directly on the device?

1.3 Thesis overview

In order to answer the research question, this thesis is divided into six main parts.
First, **the introduction to machine learning** aims to lay the foundations for a good understanding of the subject and to create the theoretical background necessary to assimilate the different mechanisms that govern machine learning. In order to reach this objective, different notions are be explained such as: artificial neural network, deep learning convolutional neural network and transfer learning.
Secondly, the second chapter discusses **the reduction of the size of a network model**. Indeed, the more layers and therefore neurons are used, the heavier the model is to store and it consumes energy. The large size of the model can be a problem when it is stored on the device. This one have limited memory, so it is important to take this parameter into account in order to make the model as small as possible. To do this, three reduction techniques are developed: model architecture, pruning and quantized model.
Thirdly, **various practical aspects for the creation of a deep learning model** for a mobile application are specified. Indeed, beyond the programming and machine learning aspects, there are other elements to be taken into account in order to carry out this type of project, such as data collection, model storage and the size of the training dataset.
Fourthly, the chapter on **guidelines for creating a CNN** aims to show and explain the different steps to create a convolutional neural network (CNN). Before writing the steps for the construction of the CNN, the choice of the library to achieve it is explained.
Fifthly, the chapter on **development of the application** introduces the different technologies to realize this. In addition, an example of an application that can be used as a template is shown. The chapter finishes with the writing of general steps for the creation of an application from scratch.
Sixthly and lastly, this thesis concludes with **the application of all the theories covered** and the scripts provided in the appendix. This part is divided into three main sections: the presentation of the study case, the explanation of the construction of the model and the different mock-ups of the application.

To validate the approach developed in this thesis, a mobile app was created. The readers interested by using a ML algorithm in an application will find in the appendix all the code needed to achieve its goal.

Chapter 2

Machine learning

This part aims to lay the foundation for a good understanding of the theme and to create the necessary theoretical background in order to assimilate the different mechanisms that govern machine learning (ML). Therefore, in order to reach this objective, an overview of ML is provided. This overview covers the basics of machine learning algorithm and explicit the differences between supervised, unsupervised and semi-supervised ML. Then, an introduction to artificial neural networks (ANN) is also made. In this part, a brief preamble concerning the history is presented, An analogy with biological neural networks, a detailed explanation of each component of an ANN model and an explanation of a classification algorithm is also made. Afterwards, deep learning is explained. Moreover, the convolutional neural network is introduced. Finally, the notion of transfer learning is discussed in order to understand the theory governing this practice.

2.1 Introduction to machine learning

Machine learning is a branch of the artificial intelligence (AI) taxonomy that is classified as a computational method based on experience and testing to increase prediction. The goal of machine learning is to use data to train a model to create a desired behavior [10]. Figure 2.1 represents the taxonomy of artificial intelligence in order to have an overview of the field.

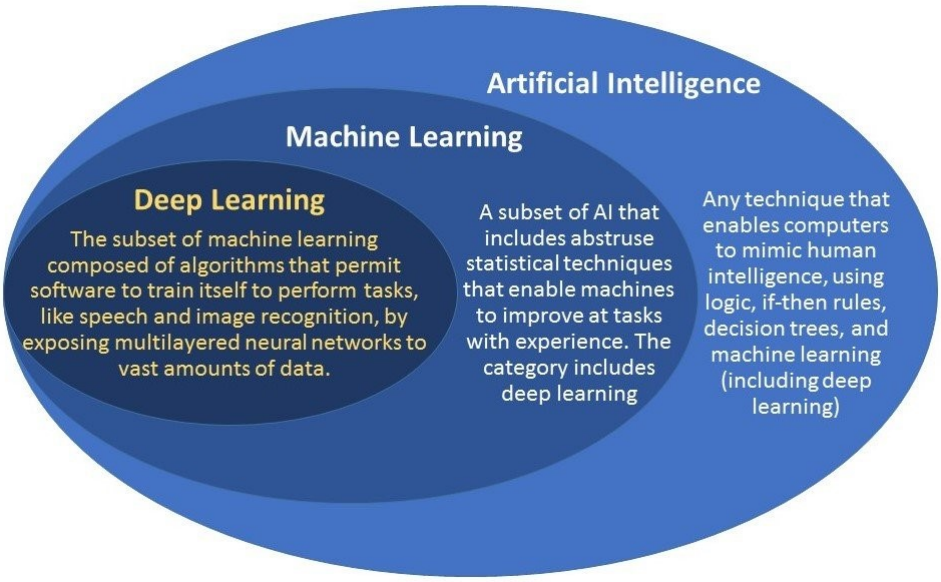

Figure 2.1: Taxonomy of machine learning [10]

For the beginning of machine learning project, it is necessary to make sure that certain requirements are met, the most important of which is the availability of a large enough number of data to be able to learn useful knowledge from them. Moreover, structured and quality data is required i.e. all data must be in the same format.
Machine learning can be defined as a way to extract information such as classifications thanks to mathematical models that used dataset to train. This one is composed of instances which are very often represented by a line of data (or a picture) of interest. To illustrate, if the model tries to classify if it is a dog or a cat, it is likely that in the dataset there is a line of data that represents a dog and that is composed of several characteristics present in different columns. This characteristics of instances are called features. For instance, these can be size, color, weight for cat and dog classification. Machine learning is divided into three sub-categories: supervised, unsupervised, semi-supervised. Below, a brief explanation is given on the three categories. However, only supervised machine learning is detailed in this thesis because it is the subject of it.

Supervised ML

The dataset used in supervised machine learning [22] are composed of the target (category) that the model tries to estimate. From then on, the model learns from each instance thanks to its features in order to create rules that allow it to predict if the instance belongs to a category or another. To do this, 2 datasets are needed: the first one is used to train the model, the second one is used to check if the model succeeds in classifying the target or not. Data present in the test dataset have obviously not been used to train the model [22]. Figure 2.2 shows an example of dataset where the objective of the model is to classify if the instance is a cat or a dog.

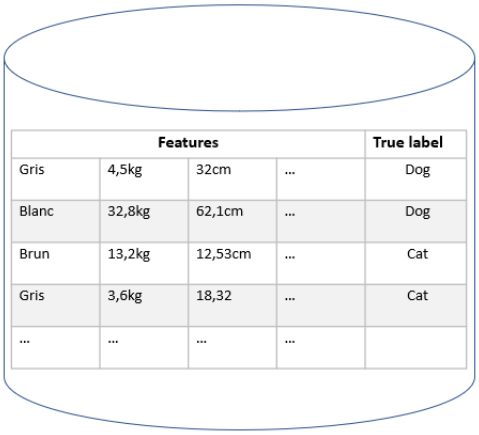

Figure 2.2: Structure of data in supervised ML

The two types of problems most frequently used in supervised analysis are regressions and classifications. The choice of the type of model is made according to the expected objective.
As far as classification is concerned, the main objective of the model is to be able to absolve characteristics in order to try to estimate, as well as possible, in which categories a certain instance is found. From a mathematical point of view, the objective of a classification is to minimize the classification error of a certain model f such as [12]:

$$\text{Error rate} = \frac{1}{n}\sum_{i=1}^{n}[f(x_i) \neq t_i] \tag{2.1}$$

where $f(x_i)$ = the estimated classification for i, t_i = the real category of i, n = the sample size.

Therefore, in general, the lower the classification error rate, the more accurate and efficient the model. There may be cases of overfitting, but this is explained later in section 5.2.5.
As far as regression is concerned, the goal is to get as close as possible to the true curve of the data. To do this, a curve is created to estimate the true curve that is unknown. There are several objective functions[1] for this type of model, but the most common is mean square error (MSE). Mathematically, this can be interpreted as follows [12]:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}[f(x_i) - t_i]^2 \tag{2.2}$$

Where $f(x_i)$ = the estimated classification for i, t_i = the real category of i, n is the sample size, e = [$f(x_i)$-t_i] = the difference between the true line and the estimated line in dot i.

In order to better visualize the mathematical logic present in this type of model which uses the MSE as an objective function. Here is an example of a linear function in two dimensions (two parameters), the objective being to reduce as much as possible the gap between the two curves (e):

[1] Objective function has the role of finding the point where the data are maximized or minimized [27]

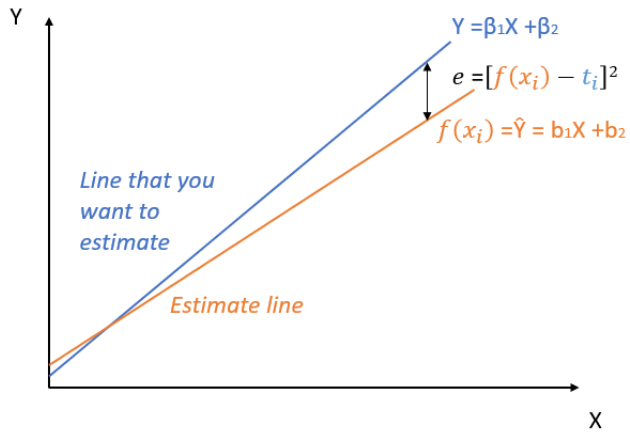

Figure 2.3: Example of a graphical representation of a regression

Unsupervised ML and semi-supervised ML

Unsupervised machine learning has the characteristic that it does not know exactly what it is looking for. Indeed, in this type of machine learning, instances in the dataset are analyzed in order to find patterns and to create groups based on similarity. In contrast to supervised machine learning, the dataset does not contain the target value (category) that the algorithm tries to recognize [35]. In unsupervised learning, clustering algorithms are the most used. The objective of clustering is to find patterns, similarities between instances in order to gather them in a group (cluster).

Semi-supervised machine learning is a category of ML that mixes supervised and unsupervised. Therefore, in this situation, some instances have the target in their features and others not. "The goal of semi-supervised learning is to understand how the combination of labeled and unlabeled data can change the learning behavior, and to design algorithms that take advantage of such a combination." [47]. Figure 2.4 shows the data structure for unsupervised ML and for semi-supervised ML:

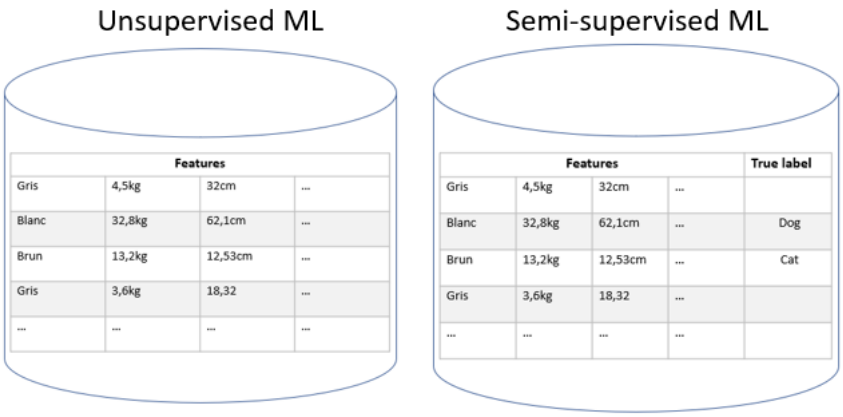

Figure 2.4: Structure of data in sem-supervised ML and unsupervised ML

After having done this overview on machine learning in order to acquire a minimum of understanding and knowledge in this field, a specialization in one of the fields of machine

learning is done. Indeed, this thesis deals with supervised machine learning and more precisely with deep learning realized thanks to neural networks.

2.2 Introduction to artificial neural network (ANN)

In this section, neural networks are explained in order to understand how they work. To do so, this part is divided into three parts: a historical explanation of ANNs, the different components of a model and the functioning of those.

2.2.1 ANNs history

Artificial neural networks (ANNs) were created in the 40's by McCulloch and Pitts. In 1949, Hebb explains, for the first time, the learning rules of neural networks [42]. The invention of ANNs comes from the will to imitate the mechanisms of the human brain. Indeed, ANNs try to simulate the electrical activity present in our brain. As it is impossible to imitate a brain due to its complexity, scientists have programmed the abstraction of its workings. To do this, ANNs imitate "the reinforcement of neuronal pathways in the brain" [42] by putting weights to these pathways. In 1957, Mr Frank Rosenblatt created the first perceptron learning algorithm. The perceptron (single layer neural network) is a supervised machine learning algorithm where a binary classification is made. In 1969, Papert and Minsky wrote and published a book (*Perceptrons*) that highlighted all the limitations of single layer neural networks. Following the publication of this book, ANNs became less and less used and researchers turn away from them. It was not until 1986 that McClelland and Rumelhart published an article on a backpropagation[2] algorithm and the addition of several layers creating the first multi-layer neural networks that the craze returned. Nevertheless, two problems remain: on the one hand, at that time, the computational power of the time did not allow to see the potential of multi-layer networks and on the other hand, the bigger and deeper a model is, the less back-propagation works.
As a result of these two problems, ANNs are still put aside [5]. It was not until the 2000s, thanks to research on unsupervised machine learning and the increase in computing power, that the field becomes a subject of interest for researchers again. However, the most important event that brought ANNs to the forefront was the annual ImageNet competition. The goal of this competition is to bring together teams that try to create a technology (AI) that best classifies images. In the 2012 edition of this competition, one team decided to use deep learning to solve this problem and their solution was twice as good as all the other teams.

2.2.2 Components

An artificial neural network is composed of neurons, layers, weights, activation function and an error function. In order to help the good understanding of this concept which is the basis for the creation of a machine learning model, an analogy with a real neural network is made. Figure 2.5 below represents the structure of a biological and artificial neural network.

[2] Backpropagation is explained in the section 2.2.3

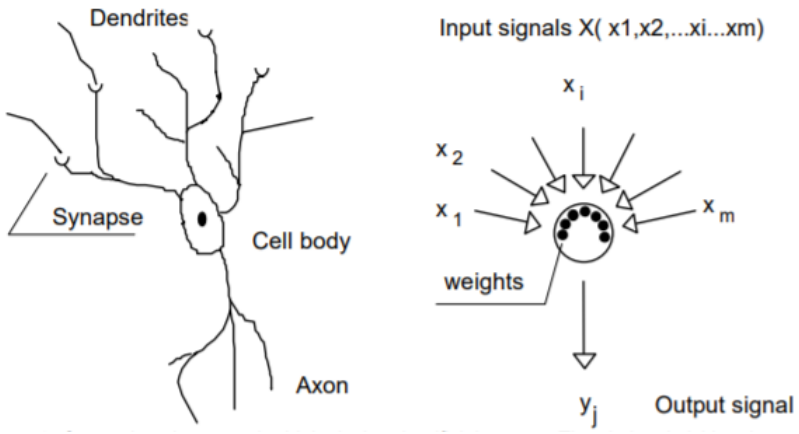

Figure 2.5: Comparison between biological and artificial neural networks [48]

First, the neuron is composed of weights and a bias that it implements [48]. These neurons are gathered in layers which are divided into several kinds: input layer, hidden layer and output layer. The input layer of neurons is the first layer of the model and it receives all instances of dataset. The hidden layers are between input layer and output layer. This one returns the probability that a data entered in the model is in a category. Secondly, the connections allow to link the neurons of different layers and can be seen as the synapses between the dendrite and the axon [48]. Thirdly, for each connection a weight is established in order to weight the importance of each relation between the different neurons. Therefore, a relation that helps classification has a high weight while a relation that does not explain a classification has a low weight.
Fourth, the activation functions, the analogy that can be made with the biological neural network is the choice of activation of certain neurons. Fifth and last, The aim of the loss function is to provide an objective. It is by minimizing this loss function that the ANN will be able to solve the problem. ANNs can be modulated in terms of components (number of layers, number of neurons...). Indeed, the structure of the ANN depends on the complexity of the classification.

2.2.3 Inner working of ANNs

Now that the model components have been introduced, it is important to understand the underlying mechanisms present in a multi-layer neural network. Below is an example of a multi-layer ANN showing the underlying calculations (see Figure 2.6). An ANN is trained with training data to recognize/classify a certain element.
To realize this training, the feed-forward method can be used. This method consists in giving random values selected in train dataset to all the input neurons present in the ANN. These values are transmitted to the other neurons but they are modified by the weights present on the different connections linking the neurons. These weights between neurons define the importance (the responsibility) that this relation has in the classification. More a relation helps in the classification, more it has a great responsibility and thus a stronger weight. From then on, the input values are multiplied by the weights of the connections and transmitted to the next layer of neurons. These values returned by the first layer of the neuron are able to be modified by an activation function which decides if it is appropriate to keep the value or not.

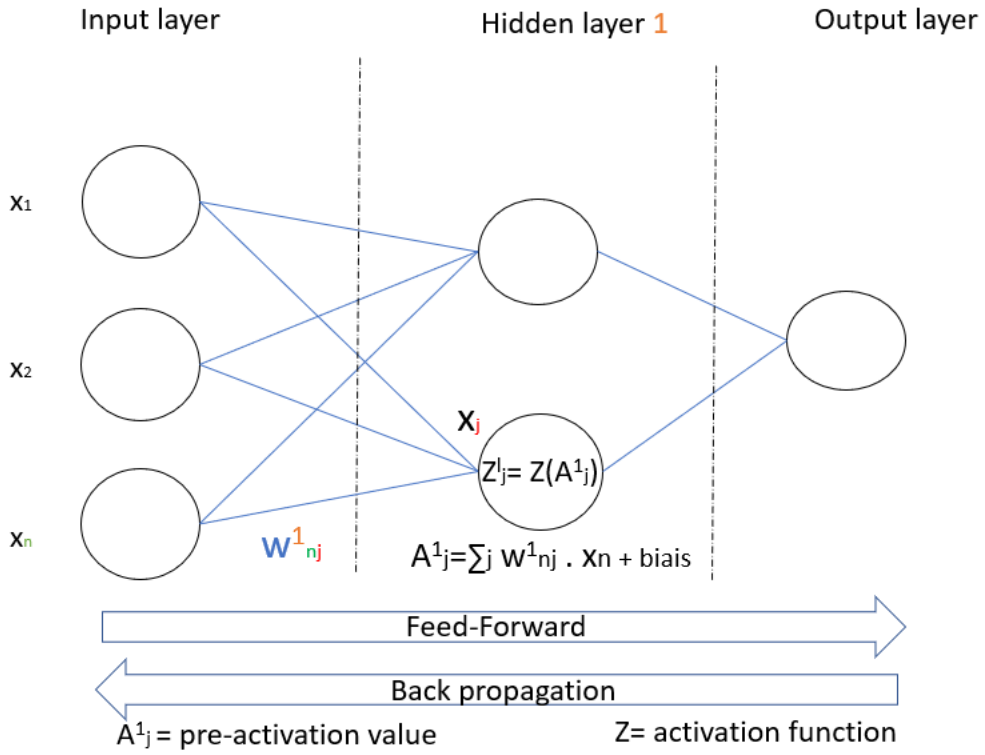

Figure 2.6: Overview ANN operation

From then on, the elements defined randomly in the first layer undergo transformations between each relation between the layers and within the neurons. This is summed and sent to the output layer. This one generally returns a probability that the input element is the object that the network is trying to classify or not. It is at this point that the error function intervenes, it estimates how much the model is wrong. For example, at the end of the training of an ANN, model says that the input are 60% of chance to belong in this category. Loss function allows to count how much model is wrong. The objective now is to modify the weights of model to minimize the error function by trying to get it as close as possible to 0. In order to find out how to modify the weights to reduce the loss function, the back propagation method is used.
This method aims at knowing how a change of weights affects the error. To do this, the model calculates the gradient descent. There are several algorithms to calculate this descent but this part focuses on the stochastic gradient descent (SGD). The choice of explaining SGD is made because it is the simplest to understand mathematically and allows to understand the intuition behind the determination of parameter values.

The gradient is the calculation of the derivative or partial derivative of a point. The gradient is a vector associated with a function and a point. For a regression, the gradient determines the coefficients of the equation (parameters) that are as close as possible to the set of points (data) of the model.

Back-propagation
From a mathematical point of view, back propagation uses the derivatives to calculate the responsibilities of each neuron based on the above equation. The adaptation performed by the neuron is calculated as follows [12]:

$$\text{adaptation} = \Delta W^1_{nj} \tag{2.3}$$

where W^1_{nj} = weight at layer 1 of the relationship between neuron n and neuron j

Gradient descent is thus an iterative algorithm that allows to find a local minimum of a differential function. Its role is to guide the learning process. The rule of gradient descent can be seen as follows:

$$W^1_{nj} \leftarrow W^1_{nj} - \alpha_t \times \frac{\delta e}{\delta W^1_{nj}} \tag{2.4}$$

where W^1_{nj} has been defined above, α_t step of the descent, $\frac{\delta e}{\delta W^1_{nj}}$ the derivative of the error with regard to w^1_{nj}

The derivative of the error with respect to w^1_{nj} is computed by each connection of each layer in order to find where to best minimize the error. In order to calculate this derivative, it is useful to use the chain rule to decompose the derivative such that [12]:

$$\frac{\delta E}{\delta W^1_{nj}} = \frac{\delta E}{\delta A^1_n} \times \frac{\delta A^1_n}{\delta W^1_{nj}} \tag{2.5}$$

where $\frac{\delta e}{\delta W^1_{nj}}$ has been defined above, $\frac{\delta E}{\delta A^1_n}$ the responsability for neuron n in layer l for error e and

$$\frac{\delta A^1_n}{\delta W^1_{nj}} = \begin{cases} Z^j_{l-1} & \text{for hidden and output neurons} \\ X_j & \text{for input layer} \end{cases}$$

derived from the pre-activation value by the weights partial derivation of the pre-activation of neuron n at layer l from the weights of neurons at layer 1 of the relationship between neuron n and neuron j.

To understand the reasoning behind this descent, it is necessary to imagine on a mountain with a lot of fog and a walker wants to go down a mountain. Because of the fog, he can't see 5 meters away. So, to find this down, He decides to move forward based on where the slope is greatest (derived from where he is). Figure 2.7 shows an application of the SDG for a simple linear regression with the loss function MSE.

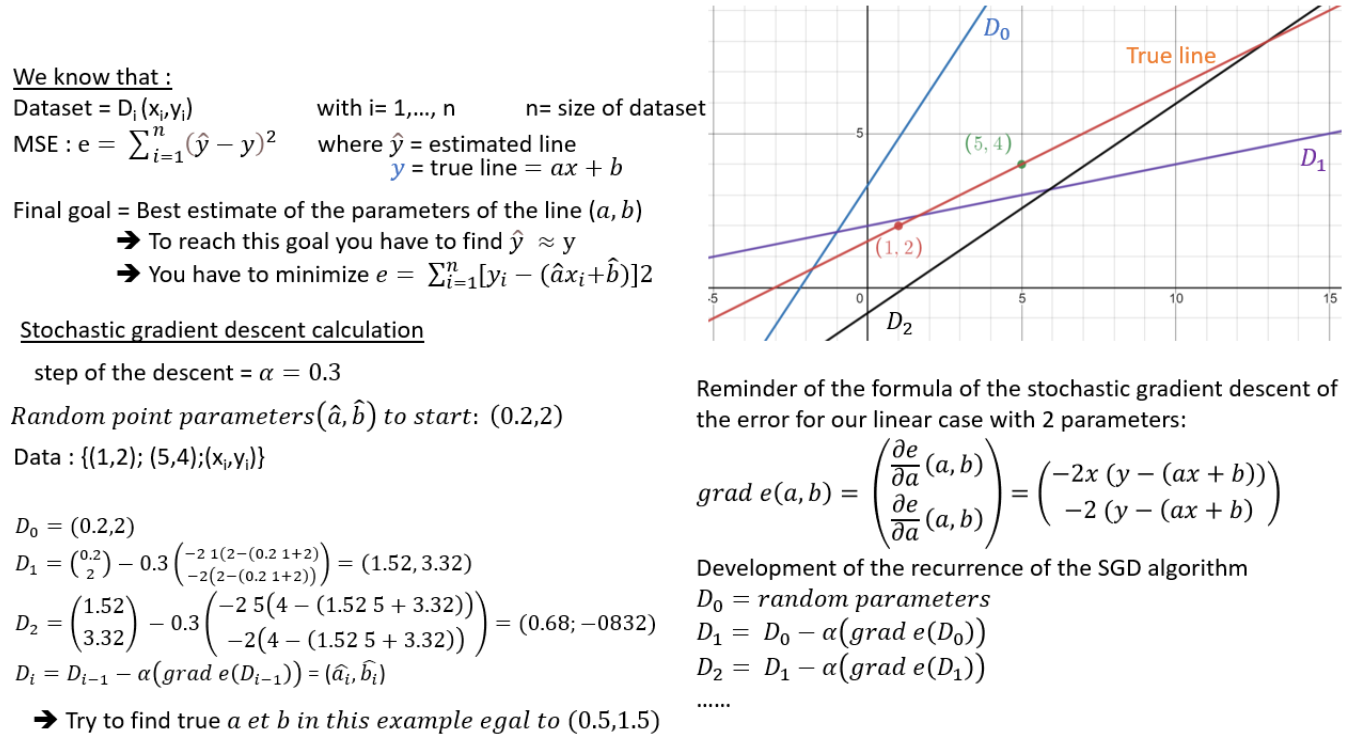

Figure 2.7: Explanation SGD

Equation 2.4 introduces the notion of the step of the gradient descent (α) also called learning rate. This one is also shown in Figure 2.7. This learning rate must be chosen arbitrarily and there is no perfect learning rate for each situation. Figure 2.8 shows the different problems encountered following a bad learning rate. For the case where α is big, this is not a problem. However, for the case where α is too big or too small it is a problem. Indeed, when α is too big, it does not manage to converge to the minimum. If α is too little, the convergence towards the minimum is done but, it requires a lot of iteration and thus of calculation what is not efficient.

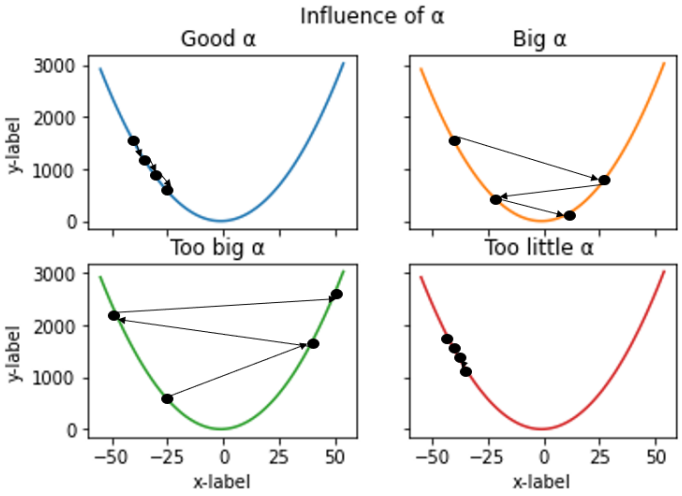

Figure 2.8: Influence of learning rate

There are several other algorithms that back propagation to improve the model. Nevertheless, this one is very often used and other algorithms are discussed later in the section 5.2.5.

2.3 Deep learning

Deep learning is a subset of machine learning as shown in Figure 2.1. Deep learning can be defined as "A class of machine learning techniques that exploit many layers of non-linear information processing for supervised or unsupervised feature extraction and transformation, and for pattern analysis and classification" [8]. Deep learning is becoming more and more popular due to the increase in the amount of data, the increase in computing power (GPU and CPU) and the progress of various machine learning algorithms. In addition, deep learning allows to classify data that are highly complex and non-linear. As for machine learning, deep learning can be divided into three subcategories which are deep networks for supervised learning, hybrid deep networks (semi-supervised) and deep networks for unsupervised or generative learning.
Deep learning is a more complicated multilayer perceptron that contains more layers. In some very complex models, it is sometimes complicated to realize the contribution of a hidden layer. Indeed, it is not always easy to understand which feature is extracted by a layer. Deep learning networks are often larger in size than classical machine learning networks. Nevertheless, this larger size has two major drawbacks. On the one hand, the increase in the number of layers significantly increases the computational power needed to compute the gradient descent. On the other hand, the increase of the computational complexity leads to an increase in the time needed to learn the model in order to have the best possible accuracy [19].

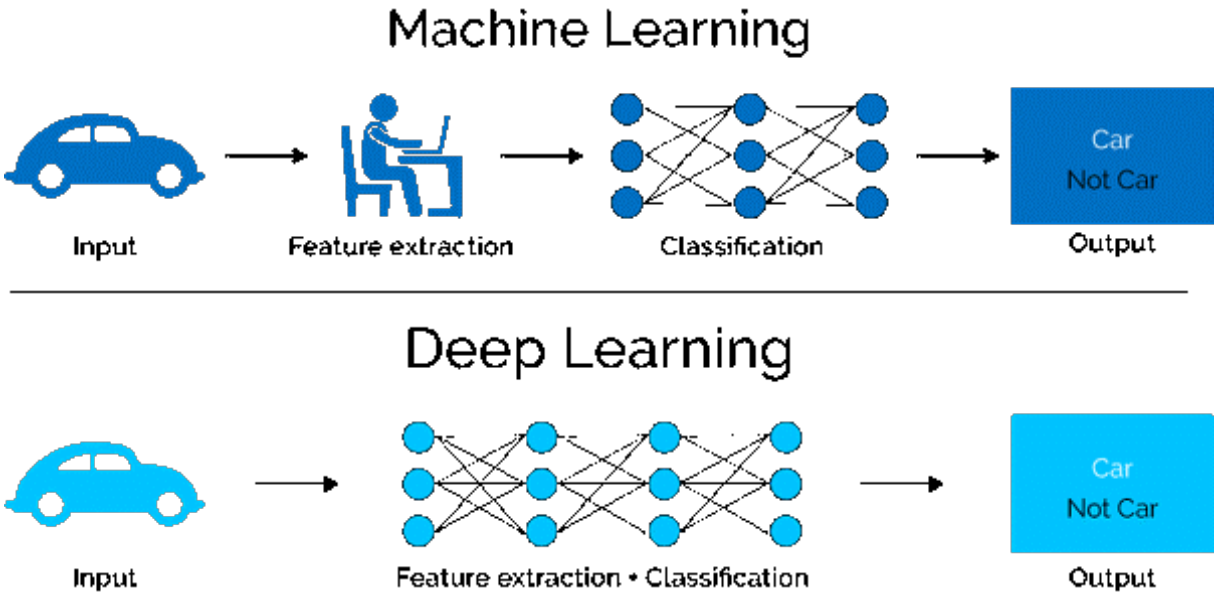

Figure 2.9: Difference between machine learning and deep learning [25]

Finally, deep learning has made it possible to perform tasks that are considered too complex for machine learning, such as automatic speech recognition. As shown Figure 2.9, deep learning includes directly the extraction of features in the model whereas in traditional machine learning, features are found usually via handcrafted pre-processing step. [25].

2.4 Convolutional neural networks

In deep learning, there is a kind of neural network that comes up often which are the convolutional neural network (CNN). This one is a type of artificial neural network that are used for image recognition [31]. Indeed, CNNs offer the advantage that they incorporate the structure of image in its model by adapting to their size and color. CNNs "extract the feature of image and convert it into lower dimension without loosing its characteristics" [36]. This specificity of CNNs allows them to obtain better results in image classification and to reduce the number calculation of a model. CNNs by their architecture allow to better manage the computational complexity than the ANNs. Indeed, ANNs have to take in parameter each pixel of image whereas CNNs try to group them in zones in order to reduce the computation costs[31]. For example, for a 64X64X3 image with colors, the input layer of ANN and CNN should contain 12,288 input neurons. Except that CNNs use processes like filters to reduce the number of computations to be performed.
In CNNs, there are 3 dimensions that are taken into account: height, width, channel. The channel can be seen as the superposition of layers caused by the colors and is equal to 3 (one layer per primary RGB color). This superposition is viewable on Figure 2.10. Each layer can be seen as a matrix where the value represents the intensity of the color for each pixel.

Figure 2.10: Overlay of color layers [36]

In conclusion, the CNNs will receive the same inputs as for the ANNs but thanks to the use of filters, they manage to reduce the calculation costs. Indeed, in CNNs instead of optimizing all the input parameters, it is necessary to optimize the filter parameters. Therefore, to continue the example of the image of dimension 64X64x3, instead of optimizing the 12888 pixels in input. The CNNS focus on the optimization of the filter. The only parameters to optimize are equal to the number of elements of the filter matrix.

2.4.1 Components

As for ANNs, CNNs are composed of neurons, layers, activation function, error function, etc. However, there are several new parameters to take into account such as filters, padding and strides.
Firstly, filters are used for two reasons : to reduce the size of the input layer and to examine the influence of the neighbouring pixels of an image [38]. The filter therefore advances from the top left to the bottom right.

Secondly, stride can be defined as the step that the filter makes. It defines how much the matrix must move pixels. For example, in Figure 2.11, the stride is equal to 2. Therefore, as indicated by the arrow, the matrix moves by 2 pixels.
Thirdly and finally, filter also takes as a parameter the padding which defines by how much the matrix present on the image (red square) can go out of the image. For example, in Figure 2.11, the matrix does not go out of the picture size. However, if the padding is equal to 1, the matrix would go out of the image by one pixel.

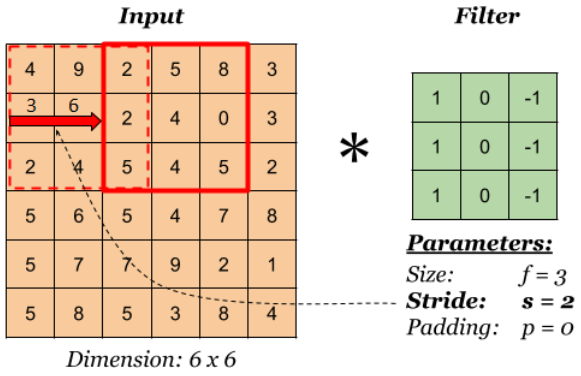

Figure 2.11: Example of padding and stride [32]

Moreover, regarding the different layers as for an ANN, there are three main kind of layers: input layer, hidden layer, output layer.
The first layer is the layer that contains the image data. These, if the image is in color, can be conceptualized as three-dimensional matrices where a matrix represents the intensity of color for a primary color (RGB).
There are several types layers in CNN: convolution, pooling, fully connected and softmax/logistic layer. In comparison with ANNs, the kinds of layers of CNNs can have different types. After the input layer, model contains a convolution layer which uses the filters. This second layer allows to extract the features of the images [30]. To do this, it uses the activation function Relu which is defined in the section 5.2.4 but which allows to set all negative values to zero. In addition, there are several types of convolution layers 1D, 2D or 3D. These define in how many dimensions the filter should work. The pooling layers can be of type max, min, average pooling. To illustrate, Figure 2.12 below shows how max-pooling works. The transformation generated by pooling gives a less good representation but it has the impact of reducing the dimension.

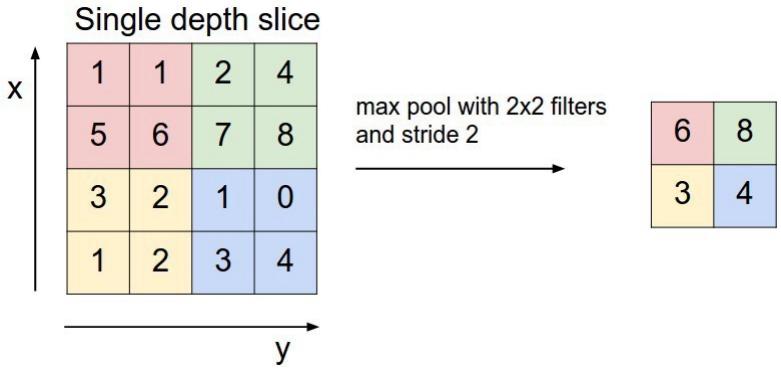

Figure 2.12: Example of max pooling [36]

The following layers use the image obtained for the first convolution and not the initial image. Another type of hidden layer is the fully connected layer which are the layers present in the ANNs. These connect all the neurons of a layer to the next layer. This type of layer is useful to sort the different images in their category [36].
Output layer as for the ANN is a vector of size equivalent to the number of categories to be estimated. The output of this layer is a vector where each place of the vector represents a category [46]. So, for example, the first value represents the probability that the input image belongs to the first category. Below is an example of a classical CNN structure where the different types of layers appear.

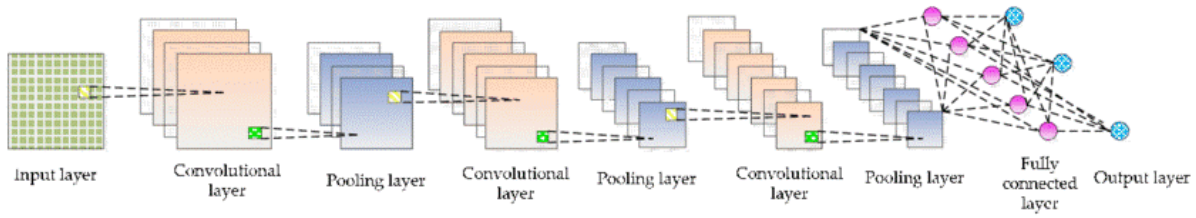

Figure 2.13: Structure basique d'un CNN [46]

2.4.2 Inner working of CNNs

CNNs work through three simple concepts [30]: local receptive fields, shared weights, and pooling.

Local receptive fields use three components explained above: strides, filters, and padding. As said before, the second layer of a CNN is not connected to each image pixel but to a part of pixels called "local receptive field". This one represents a small region of the input neurons [30]. This hidden layer is the convolution layer and it takes an overall bias. Moreover, local receptive field passes over the whole image (input) and each neuron of a layer represents a precise area of the image defined by the local receptive field.

The shared weights is a mechanism that gives for each neuron of a same layer the same weights. More precisely, each layer has a different filter but it is the same for each neuron of the layer. Each layer of neurons has its own filter. This filter will be applied everywhere in the same way for each neuron. So that the extraction produced by the filter is done in a homogeneous way on the whole image. This mechanism of shared weights in a layer allows to detect exactly the same features everywhere in the input image as each neuron of the layer focuses on a part of the image. Therefore, each layer of the CNN tries to create an abstract representation of the image. Generally, the first layer learn to focus on the luminance present in the image and for example, the second layer could be in charge of a precise feature like a physical characteristic (finding if there are eyes in the image).

The last concept is pooling which has already been introduced in the previous section. Nevertheless, it is important to understand why it is used. Pooling allows to simplify the information of the output generated by the convolution layer. This is why a pooling layer is usually found after a convolution layer [30].

Figure 2.14 shows that the first layers focus more on the geometrical aspect but the more the layers advance and the more layers try to understand semantics existing behind image. To do so, the network assigns to each layer a ”task” to try to extract a particular feature. Figure shows the feature extraction by three different CNNs : AlexNet,VGG-19 and ResNet-50.

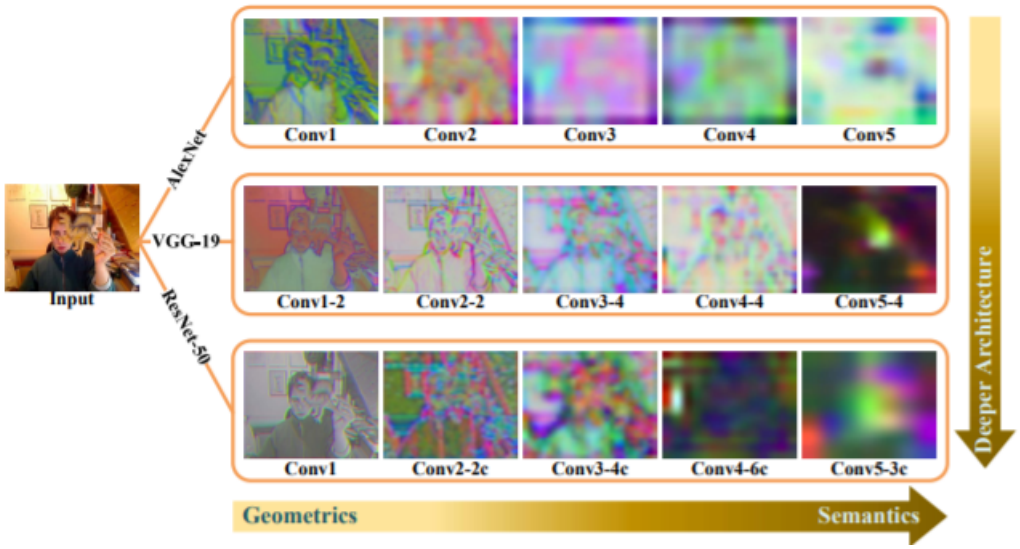

Figure 2.14: Feature extraction by layer for 3 models [14]

The back propagation method explained for ANNs also works with CNNs. As a reminder, this method is explanation in section 2.2.3. Nevertheless, the objective of CNNs is to improve the convolution matrix in order to extract the best features from an image. The role of the improvement of this convolution matrix is to create the best semantic representation of the image, i.e. they try to calculate the impact of different areas of the image to classify [46]. For example, to classify men and women, as women usually have long hair and men have short hair, the extraction of this feature on the image is a good indicator to define if it is a man or a woman.

In conclusion, CNNs show many advantages for image classification. CNNs are often very efficient when classification is subject to local correlations to help classification [30]. Indeed, CNNs highlight important places to look and features to have.

2.5 Transfer learning

Following effervescence of machine learning and more precisely, deep learning, transfer learning is more and more used. This field of research that transfer learning represents can be defined as ”an important tool in machine learning to solve the basic problem of insufficient training data. It tries to transfer the knowledge from the source domain to the target domain by relaxing the assumption that the training data ”[39]. There are two ”types” of transfer learning : data oriented or network oriented.
On the one hand, The first type of transfer learning focuses on data transfer. This means that data from another model is used to train a new model. In data oriented, there are two sub-categories which are: mapping-based and instance-based. Mapping-based aims to create a new dataset based on the source and target domains. instance-based assumes that, despite the difference

in domain between the source and the target, there are specific instances in the source domain that, with proper weighting, can lead to an increase in the size and validity of the training dataset [39]. Figure 2.15 shows the representation of the data transfer in the two subcategories.

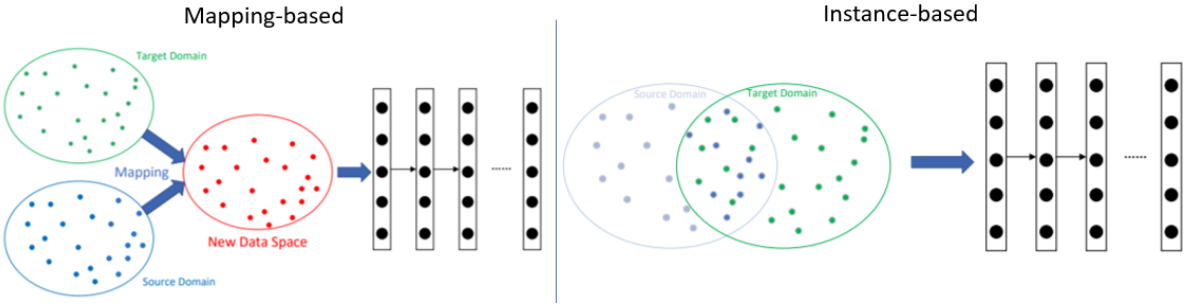

Figure 2.15: Sketch map of mapping based and instance based [39]

On the other hand, network-oriented transfer learning transfers the model itself. This means that a network created for another problem can be reused. There are also two subcategories of network-based learning (network-based, adversial-based) but only network-based is discussed. The choice to explain only one subcategory comes from the fact that network based is most often used in practice and the second subcategory requires deep knowledge in the ML domain. Network based transfer learning aims at reusing the pre-training on a model to reuse it on another domain. The different neurons, layers as well as their weights, connections and more generally the structure are reused. The only modification to provide when using this category is the change of the last layer of the model to adapt it to the category numbers of the targeted domain.

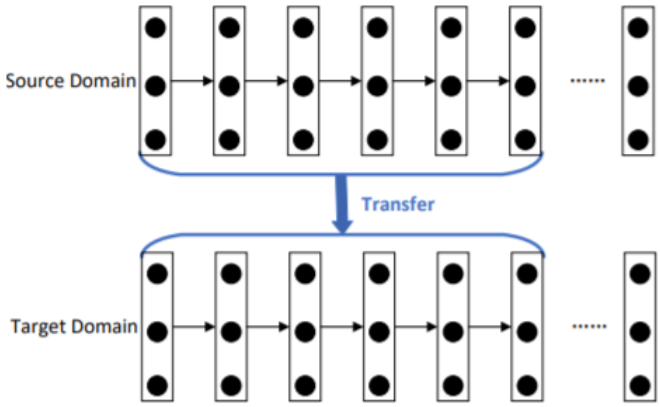

Figure 2.16: Sketch map of network-based deep transfer learning [39]

To conclude, machine learning allows to save time in the creation of the model because it reuses an already trained model and it allows to avoid heavy calculations while learning the model. Therefore, before starting the creation of a model from scratch it is interesting to consider the creation of a model thanks to transfer learning.

To keep in mind from this chapter

1. There are 3 types of machine learning (supervised, semi-supervised, unsupervised)
2. About ANNs:
   - ☞ Components: neurons, weights, activation function ...
   - ☞ Use of a back-propagation algorithm to improve the accuracy of the model by adjusting the parameters
3. Deep learning are larger and heavier neural networks:
   - ☞ Deep learning includes directly the extraction of features
   - ☞ deep learning allows to classify data that are highly complex and non-linear.
4. About CNNs (ANN specialized for image classification):
   - ☞ Components: neurons, weights, activation function, filters, padding, strides ...
   - ☞ Different types of layers: convolution, pooling ...
   - ☞ Objective: modify filters to extract the best possible features
5. Transfer learning is the use of data or model to create another model:
   - ☞ 2 types of transfer learning: data oriented (mapping-based and instance-based) or network oriented (network based)

Chapter 3

Reduction of the model size

This chapter talks about the reduction of the size of the convolutional neural networks model. Indeed, The more layers a model contains, the heavier it is. the heavier the model is, the harder it is to store it into device. Moreover, energy consumption is proportional to the size of the model. Cell phones have limited memory, so it is important to take this parameter into account in order to make the model as small as possible. In order to ensure that everyone can download application, it is necessary to try to create the lightest application possible and therefore reduce the size of the model while maintaining a good level of accuracy. Currently, there are three main solutions to limit the size of the model. First, the use of networks (CNN) designed for mobile. These are smaller and require less computing power to do the classification. Secondly, it is possible to cut the model to reduce its size (pruning). The third and last solution is to use a framework to quantize the layers of the model.

3.1 Network architecture for mobile

There are neural networks that are specifically designed to be used on a device. In this type of network, it is important to find the right trade-off between model size and model accuracy. Indeed, the physical limitations of the device constrain the possibilities in creating the model. On the one hand, the model cannot be too heavy because the device cannot support complex calculations that require a lot of energy. Therefore, model must be small enough so that the network works on the phone. On the other hand, despite this limit in the size of the model, it is still important to have a model with a fairly high accuracy rate.
There are already many proven CNN models that can be reused thanks to the transfer learning explained in the section 2.5. This is why it is interesting to talk about how these CNN are designed in order to decrease their size while keeping a high degree of accuracy. The goal of this part is to discuss some techniques used to achieve a good trade-off between size and quality. These different techniques could be used on others models. Among the possible techniques, two are chosen and presented from two existing models in order to see what mechanisms they use to decrease their size while keeping a high level of prediction.
For the first version of MobileNet (2017), the idea is to replace layers that require heavier computations with smaller ones. To do this, the high-dimensional layers are decreased. For example, by replacing "expensive convolution layers by a cheaper depthwise separable convolution[1], which is a 3×3 depthwise convolution layer followed by a 1×1 conv layer" [17]. This is less computationally expensive but does pretty much the same thing.
For MobilenetV2 (2018), this is the same logic as for the 2017 MobileNet but the layers have been rearranged. In this configuration, the deepest layer is now in the center of the model. Before and after this layer, there is a convolution layer of dimension 1X1. To do this, the filter is divided in two sub-matrix. Figure 3.1 shows an example of separable convolution with a filter of dimension 3X3. In this Figure, filter is divided into 2 sub-filters which are of dimension 3X1 and 1X3 [43] . In the case of MobileNetV2, the two layers placed between the deep layer have two distinct roles. "The first layer is called a depthwise convolution, it performs lightweight filtering by applying a single convolutional filter per input channel. The second layer is a 1 × 1 convolution, called a pointwise convolution, which is responsible for building new features through computing linear combinations of the input channels" [34].

$$\begin{bmatrix} 3 & 6 & 9 \\ 4 & 8 & 12 \\ 5 & 10 & 15 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

Figure 3.1: Example separable convolution [43]

[1]The mechanism to reduce the computation is called "depthwise separable convolution" [34]

These two examples show the impact that architecture can have on the efficiency of a model. These computation reduction methods can be established for others models.

3.2 Cutting model (pruning)

Pruning is a very common method in machine learning. It aims at eliminating individual parameters or group parameters (like a neuron) to make the model more efficient [3]. A model is considered more efficient when its size decreases but its accuracy rate does not change. There are several different approaches to pruning. Indeed, some consider that pruning should be done periodically during the model [13] or that it should be initiated directly when the model is created [24]. In this thesis, it is not a question of knowing approaches is the best that it goes out of the scope of the latter.

Pruning can be seen as a process as shown in Figure 3.2 [29]. The first step is to calculate the importance of each neuron. In the second step, the neurons considered as less useful are removed. A neuron is considered as not very useful when it brings little in the extraction of features from an image and/or when it does not significantly increase the accuracy rate of the model. To find these less performing neurons, the model is run with and without the neuron to see the delta of the accuracy rate between the two. The third step of the model is fine-tuning, i.e. re-running the code so that the model trains again without the non-efficient neurons. The last step consists in iterating in order to obtain the expected trade-off between decreasing the size of the model and maintaining the level of accuracy.

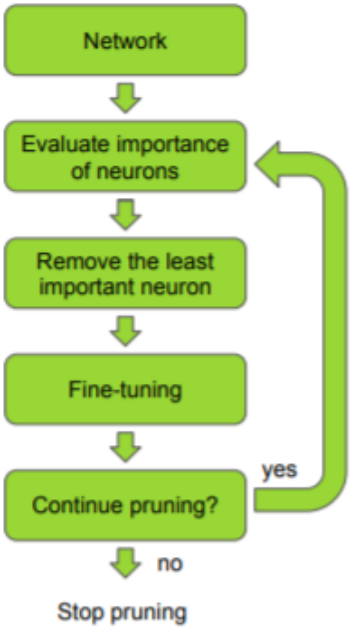

Figure 3.2: Process for pruning [29]

The goal of pruning is to reduce the number of parameters under the constraint that the number of parameters remains greater than zero. In addition to this, there is the desire to keep the accuracy of the model with all parameters the same. Model after pruning becomes the same model with a subset of initials paramaters.

To conclude, pruning is therefore a good method to reduce the size of the model. However, contrary to the first solution, it requires better computer and programming skills to be implemented.

3.3 Improving the efficiency of a model

The last solution is the improvement of the model itself. This solution is more difficult to put into practice than the first two because it requires a deeper knowledge in the field of machine learning and computer science in general. Nevertheless, in some cases, the first two may not work and yet it is necessary to improve the efficiency of the model.
The article *Quantized Convolutional Neural Networks for Mobile Devices* proposes " an efficient framework, namely Quantized CNN, to simultaneously speed-up the computation and reduce the storage and memory overhead of CNN models" [45]. The idea is to quantize some types of layers to improve the efficiency of the model.
In this work, the principle of this framework is briefly explained and only for the full connected layer (principle represented in Figure 3.3). Therefore, if this method is interesting, reading this article is recommended in order to understand the logic of the quantized convolutional neural network.
To quantize a full connected layer, the weight matrix of the layer is decomposed into several subsets (sub-vector splitting). The weight matrix is decomposed into M sub-matrices and the input data can be learned in M subspaces. The decomposition of the matrix size reduces the computational complexity. The dot product of the input layer sub-vectors can be done with the weight matrix sub-vectors. This decomposition of the product calculation reduces the computational time. The product of each sub-vector of the input and the sub-vector of the weight matrix is stored in a "look-up table". Then, an addition of all the products is done and it gives the response of layer. This decomposition reduces the complexity of the calculation. Indeed, according to the authors, it goes from $O(Cs * Ct)$ to $O(CsK + CtM)$ where cs and ct are respectively the dimension of the input and output layer and k and m are the number of subspaces respectively for the input layer and the weight matrix.

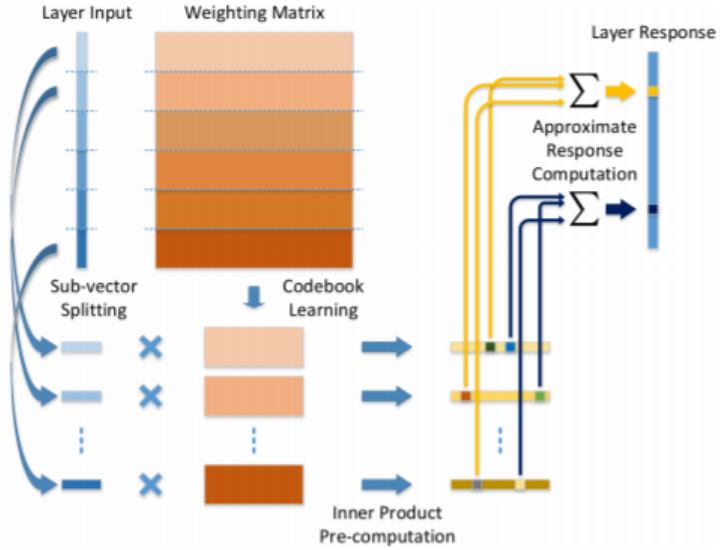

Figure 3.3: Process of quantization for a full connected layer [45]

This solution is more complicated but it allows to reduce the size of a model in case you want to create model from scratch and it is too big to have a normal use on a device. Before trying this solution, it is easier to perform a pruning which is a more widespread method to manage this kind of problem.

To keep in mind from this chapter

Three process to reduce model's size:

1. Use of networks designed for devices or little networks
   - ☞ Use transfer learning to get these models
   - ☞ If the model is created from scratch, take inspiration from mechanisms implemented in these models to reduce the size (split filter, change the order of the layers, . . .)
2. Using pruning to cut the model
   - ☞ This process eliminates the least important parameters
   - ☞ Requires more programming notions
3. Use of a framework to quantize some computationally intensive layers of the model
   - ☞ This framework requires good knowledge in programming and in CNN
   - ☞ Useful when the first two did not work

Chapter 4

Practical aspects

This chapter discusses various practical aspects for the realization of an application using deep learning. Indeed, beyond the programming and machine learning aspects, there are other elements to take into account to realize this type of project. Therefore, this chapter is divided into three parts: an explanation of different methods to collect data, the issue of storing the model is addressed, the use of techniques to increase the number of images.

4.1 Data collection

To create a deep learning model, a lot of data is needed, and these data are not always easy to find. For each category that a model estimates, a large number of training images are needed in order that model learns. In addition to this, the test data must be different from the training data. There are 2 main ways to retrieve data in order not to take all the pictures by itself : website, collecting data via a form. It is important to keep in mind that it is the quantity and quality of the data that ensures the creation of a successful model.

Website

There are specialized websites where data are free of charge like Kaggle or ImageNet. The goal of these websites is to give free access to data to be used to train, test and validate neural networks or other. ImageNet is created and managed by Stanford University and Princeton University in the United States. The disadvantage of these websites is that there is obviously no dataset for each object. In addition, for website it's really important to check the licenses. Indeed, on internet, there are many image classification projects where the authors have given free access to the data they have collected to make their model. These can also be a great source of data for model creation. To know when the data can be used or not, there are several licenses that indicate to what extent the data, or even the code, can be used. There are 5 types of software license models : Public domain, Permissive, LGPL, Copyleft, Proprietaty [40].
Firstly, the Public domain is the most permissive license. In fact, everybody can modify and used the resource. Secondly, the Permissive contains few restrictions to using. With this licenses, it's possible to modify and redistribute the code. A example of permissive licenses is MIT license [2]. Thirdly, LGPL (GNU Lesser General Public License) allows to link a open sources libraries to a project. However, any changes made to a code with such a license must release the application under similar terms as the LGPL. Fourthly, Copyleft allows the reuse of resources but prevents the possible evolution of a work from being accompanied by a restriction of rights. In other words, this license shares resources as a common good. Lastly, Proprietary is the most restrictive licenses. It means that all rights are reserved. In the case of creation of a new model, it's interesting to have the two first types of software licenses model.

Harvest thanks to a form

Another possibility is to do a participative data collection. Indeed, a survey via a Google Forms can be set up to collect data. This method allows to collect data that are not on a website. The advantage of this method is that it allows to reach faster the number of images required to train model. However, the drawback of this method is that the images belong to the person who took the picture. It is therefore necessary to ensure compliance with the General Data Protection Regulation (GDPR) in order to use these images. To be able to use the collected images, it is necessary to ask the author's agreement in order to make sure that he agrees to the use of these images. In appendix .1 there is a small legal text that allows to ask the author of the image to use these images. This text can be used for Belgium and it is to be put at the beginning of the form or others platforms to collect data.

After data collection, it is important to check the structure of the data. All of the data collected must be preprocessed i.e., see if it seems coherent, classify it if it is not, and verify that all the data is structured in the same way.

4.2 Model storage

For the storage of the CNN, there are two main possibilities: either storing the model directly on the device, or storing it on a cloud.
On the one hand, for storage on the cloud, it works on the principle that the device sends a request to the web service and this one returns the result of the prediction [16]. Figure 4.1 shows the process when the model is not stored on the device. This choice offers the advantage that the size of the model (a key factor for this type of project) is no longer a constraint for this project. There is no need to try to create the most efficient model possible. Moreover, it is easier to subcontract with a company to use an API that would take care of the model management. However, this choice brings some inconveniences such as the impossibility to run the application off the network. In addition, one must be careful about data protection. In fact, the photo that a user takes with the application is no longer only on the phone. Transferring this image to the cloud can be problematic in terms of privacy.

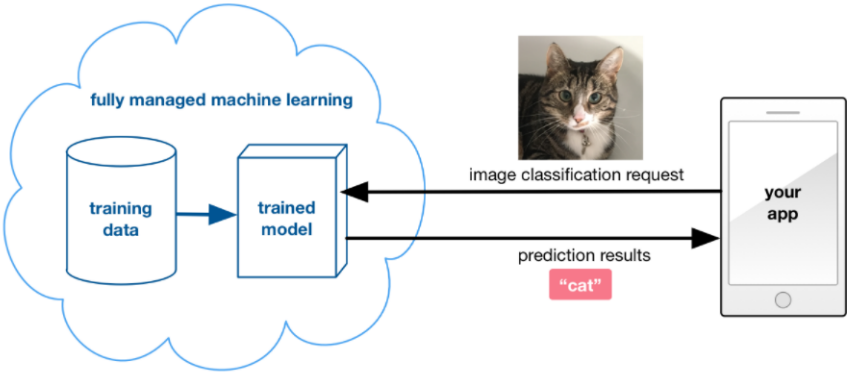

Figure 4.1: Overview of transferring an image to the cloud [16]

On the other hand, the other possibility is to store the model directly on the device. This solution requires attention to the size of the model and the power needed to run the model. Storing the model directly on the device allows the user to use the application without the need for internet and the images taken by the user never leave the device, which ensures compliance with the GDPR.

4.3 Data augmentation

In some cases, there may be too little data available to train a model. A lack of data can induce the creation of a model with a not good enough level of accuracy because it does not have enough data to train properly and absolve classification rules. To overcome this deficiency, there are different techniques to increase the number of images present in the dataset such as using data-oriented transfer learning. Another one is data augmentation which consists of in the modification of some data (picture for a CNN) of the training dataset to increase its size. Data augmentation is always used for training datasets and not for test and validation datasets. The exclusion of data augmentation in these two datasets is explained by the desire that pictures should represent as much as possible images of reality. In real life, there are no images that have ”suffered” data augmentation. There are two approaches to data augmentation: data warping

and synthetic over-sampling [44].

On the one hand, data warping deforms the images in a random way which generates additional samples [33]. The deformations can be of several types: translation, relation, shearing. From then on, the transformation of the images allows to add additional samples. A python script to do data warping can be found at appendix .2.

On the other hand, synthetic over-sampling is a solution that operates in "feature space" instead of "data space" as in data warping [33]. This technique is called SMOTE for "Synthetic Minority Oversampling Technique". It is used when there is an imbalance in terms of numbers in the different categories [4]. This imbalance causes a problem in classification because the underrepresented class cannot correctly extract "rules" to properly learn to recognize images. The figure shows that there are more of one class than the other (more green dots than red triangles). In this situation, SMOTE can be used. It works by using the k nearest minority class neighbors (k-means)[1] which will allow to create data that are close in terms of features to complete the class that is under represented [4]. In the figure below, the k-means allowed to define where to recreate data and the data created in addition are the blue triangles.

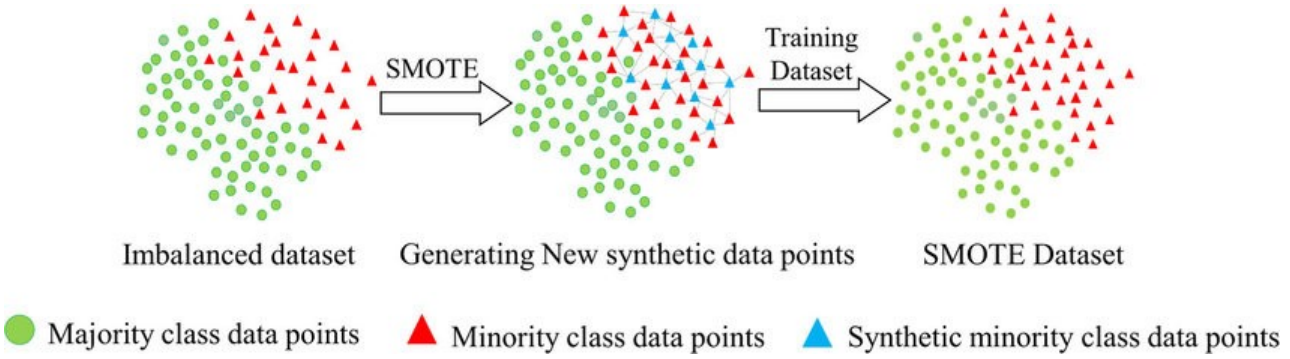

Figure 4.2: Evolution of the dataset after the use of SMOTE [1]

[1] K-means is a clustering algorithm that works on the principle of finding the "average" data point that is in the center of a cluster (category). To do this, first, the algorithm designates random points as the center of a cluster (the number of points being equal to the number of clusters). Each other data point is assigned to the center point that is closest to it, forming the first clusters. Then, for each cluster, the point closest in average to the others is defined as the new center point. Finally, this mechanism is repeated until a clear distance between the clusters is established

To keep in mind from this chapter

1. Data collection
   - ☞ Use of websites such as ImageNet or Kaggle
   - ☞ Participation of other people to build the dataset thanks to a form
2. Model storage
   - ☞ Cloud
     - Advantages : subcontracting of the model creation, no constraints on size
     - Disadvantages : restriction due to the GDPR, obligation to be connected to the internet
   - ☞ Device
     - Advantages : Respect of the GDPR, no obligation of a connection to Internet
     - Disadvantages: Adaptation of the architecture of the model for a mobile
3. Data augmentation
   - ☞ Data warping
     - Modification of images to create new ones
     - Creation of an image based on the data dimension
   - ☞ Synthetic over-sampling
     - Useful when a category is under represented
     - Image creation based on the feature dimension

Chapter 5

Guidelines for CNN creation

This chapter aims to show and explain the different steps to create a convolutional neural network (CNN). The choice of the framework for the creation of the model is introduced. The different steps to create the model are explained as well as the different points of theory for the realization of the steps. Moreover, scripts for creating a model from transfer learning and data stored on a drive/disk is directly in the thesis and the script for model creating from scratch is between appendix .3 and .6. The choice to incorporate the scripts for transfer learning directly into the text was made because it is the easiest way to create a model and it is the most used method. The steps for creating the CNN are: downloading the libraries, downloading the data, exploring the data, creating the model, compiling and running the model, evaluating the model, saving and loading the model, and converting to a Tflite model.

5.1 Pytorch or Tensorflow libraries

There are many libraries to create CNNs but the two most used are Tensorflow and Pytorch. Both are open source frameworks created respectively by Google and Facebook. There are several major differences between these two libraries.First of all, the graphics for Pytorch are dynamic whereas they are static for Tensorflow. This means that with Pytorch, it is possible to manipulate the graphs that are being executed [20]. Then, as far as deployment is concerned, Tensorflow has a framework dedicated to that which is called Tensorflow Serving[1]. On the contrary, Pytorch does not have its own framework to deploy itself and so it requires the developper to expose the model through a custom REST API[2] [11]. Afterwards, Tensorflow has a larger community than Pytorch [20] because it is a library that has been around longer. This does not prevent Pytorch from catching up more and more people and to increase the number of ressources an tutorial for its library. Moreover, Tensorflow offers a tool called Tensorboard which allows to visualize the evolution of the model. Unfortunately, Pytorch does not have this kind of service and requires another library like Matplotlib [20].

In conclusion, both libraries have their qualities and their defects and it is difficult to say which one is the best. Moreover, all advantages and disadvantages have not been quoted. Nevertheless, for this thesis, it was decided to use Tensorflow because its community is larger and its literature better written.

5.2 Steps to create model

This section covers all the steps necessary to fully train a model using the Tensorflow library. The first step is to create a model using the Tensorflow library.

5.2.1 Libraries importations

To start the creation of the model, it's necessary to import the different libraries. Tensorflow and Keras can cause problems during importation. Indeed, these two libraries are strongly subject to versioning problems. However, a more stable version has been released in January 2021. The most used libraries for this kind of project are : Tensorflow, Matplotlib, Pandas, Numpy and OS. Each one has its own specificity.First, Tensorflow, as discussed above, is the library that allows to create the model itself. It is a specialized library for machine learning and especially for deep learning. Secondly, Tensorflow contains Keras. It is also a machine learning library but it is higher level than Tensorflow. Indeed, the back-end behind Keras is Tensorflow. The real difference between the two is that Tensorflow allows to make very precise changes and it is not possible with Keras. In general, when a model is created, both libraries are used. Thirdly, Matplotlib library allows to create visuals such as graphics. For example, this library allows to display the evolution of the precision rate or to display the different images during the image exploration stage. Fourth, Pandas is a library for the manipulation and analysis of raw data. Indeed, Pandas has developed DataFrame which serve as a large array with columns and rows to store data. Fifth, Numpy is also a library for data manipulation. In this case, Numpy

[1] https://www.tensorflow.org/tfx/serving/serving$_$basic
[2] https://pytorch.org/tutorials/intermediate/flask_rest_api_tutorial.html

is essentially used for matrix and/or vector manipulations. As a reminder, images are large matrices where each pixel represents an element of the matrix. Lastly, the OS library is used to manipulate the folders and files present on the computer. It is also used to save the model. Code 5.1 shows importations of all these libraries.

Source Code 5.1: Libraries importations

```
import tensorflow as tf
import keras
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import os
```

5.2.2 Data downloading

The real first step after importing the different libraries is to download the data on the computer. There are several ways to do this.

Data from libraries

On the one hand, it is possible to download data manually but most of the time popular datasets are provided by libraries. So it is enough to call a function to download dataset. Appendix .3 shows an example of downloading a "MNIST" dataset using Tensorflow and Keras.

Data from drive/computer

On the other hand, another possibility to load these data is to get them directly on drive or on pc. This solution is used when there is no existing open source dataset. Code 5.2 below contains a function that allows to see how many images each category has and to download the data from a drive to the IDE. In addition, it returns two directoryGenerators (train and test) that are used during model creation. In order to facilitate this step, a function has been created with the following parameters: path_ data_ train, path_ data_test, shuffle, size_ of_ image, color_ mode and class_ mode. The first two are used to indicate where the training and test data are stored. For the other parameters, these are parameters for the creation of a directoryGenerator (object used for the creation of the model). To do this, it is necessary to use the Tensorflow function "train_ datagen.flow_ from_ directory()". From then on, the other parameters of the load_ your_ data() function are parameters of the Tensorflow function. There are obviously other parameters for this function that are available on their site [3]. The other four parameters in the load_ your_ data function already allow to modulate the function to data. Indeed, the thirdly parameter defines the size of images (dimensions). The fourth parameter indicates whether the image is in color or not. If image is in color, the parameter has to be equals to "rgb" and if image is in black and white, the parameter has to be equals to "grayscale". The fifth parameter allows to shuffle data which is highly recommended for learning. In fact, it is important to shuffle data to avoid that model anticipates the next image and classifies it in the same category as the previous one. The sixth parameter is used to determine the type of label (category name). For example, if it is a classification between two categories, the parameter class_mode has to be "binary" and if the labels are exclusive, paramater has to be "categorical". Exclusive label means that the

[3] https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator#flow_from_directory

same image cannot be in two categories at the same time. There are other class_ modes that are available in documentation (same link that before).

Source Code 5.2: Function load your data from drive/hardware

```
def load_your_data(path_data_train,path_data_test,
    size_of_image=224,color_mode='rgb',shuffle=True,
        class_mode="categorical"):
    """
    Function to load data and to return the directory to
 ↪  create model

    :param path_data_train: path where your training data is
 ↪  stored (str)
    :param path_data_test: path where your test data is
 ↪  stored (str)
    Optional parameters
    :param size_of_image: hape of your image  (int)
    :param color_mode: "rgb" if image in color, "grayscale"
 ↪  if black and white (str)
    :param shuffle: Shuffle your training data (Boolean)
    :param class_mode: Determines the type of label arrays
 ↪  that are returned "categorical"
    if This is a mutually exclusive labels. "binary" if two
 ↪  classes, see docs for more details (string)

    :return: return 2 directoryGenerators that uses for
 ↪  create your model
    """
    import os
    from keras.applications.mobilenet import
     ↪  preprocess_input
    from keras.preprocessing.image import ImageDataGenerator
    CATEGORIES = os.listdir(path_data_train)
    listdir = []
    for category in CATEGORIES:
            path_train = os.path.join(path_data_train,
             ↪  category)  # concatenation
            path_test = os.path.join(path_data_test,
             ↪  category)  # concatenation
            class_num = CATEGORIES.index(category)
            count = 0
            for img in os.listdir(path_train):
                    count += 1
            print('Category :', category, ' Number of images
             ↪  :', count, "Dataset : training")
            count = 0
            for img in os.listdir(path_test):
```

```
                    count += 1
            print('Category :', category, ' Number of images
             ↪  :', count, "Dataset : test", )
    # same goal that 1/255 to rescale
    train_datagen =ImageDataGenerator(
    preprocessing_function=preprocess_input)
    test_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input)

    train_generator = train_datagen.flow_from_directory(
    path_data_train,target_size=(size_of_image,
     ↪  size_of_image),
    color_mode=color_mode,batch_size=32,
    class_mode=class_mode,shuffle=shuffle)

    test_generator = test_datagen.flow_from_directory(
    path_data_test,target_size=(size_of_image,
     ↪  size_of_image),color_mode=color_mode,batch_size=32,
    class_mode=class_mode,shuffle=False)

    return train_generator, test_generator<
train_generator, test_generator
 ↪  =load_your_data('path_for_training_data','path_for_test_data')
```

5.2.3 Data exploration

The third step is to check that the data has been loaded into model and that it has not been modified. This step is not mandatory in the sense that it does not affect the creation of the model. Nevertheless, it is always important to the data before starting the training. This function in Code 5.3 returns random images of each class. Following the return of this function, it is possible to check if data appear correctly and if they are in the right category. This function only takes as parameter the path to access the training or test data.

Source Code 5.3: Function for data exploration

```
1  def data_exploration(path_data_train):
2      """
3      Show random image and their categories
4
5      param : path where trainind data is stored (string)
6      return image of dataset
7      """
8      import os
9      import matplotlib.pyplot as plt
10     import random
11     import imageio
12     class_names = os.listdir(path_data_train)
13     count=0
```

```
  full_list_path_img=[]
  for name in class_names :
    count+=1
    class_path = str(path_data_train)+'/'+name
    list_path_img = os.listdir(class_path)
    for i in range(5):
      img_path =random.choice(list_path_img)
      full_list_path_img.append(class_path+'/'+img_path)
  count=0
  fig = plt.figure(figsize=(20, 20))
  for img in full_list_path_img :
    count+=1
    plt.subplot(len(class_names),len(class_names),count)
    plt.xticks([])
    plt.yticks([])
    title = os.path.basename(img)
    plt.title(title)
    img = imageio.imread(img)
    shape = str(img.shape)
    imgplot = plt.imshow(img)

  return plt.show()
data_exploration("your_data_train_path")
```

5.2.4 Model creation

Step four is the key step in the creation of a CNN because it is the model creation step. There are two main ways to create a model: either built it from scratch, or built it from transfer learning (see section 2.5). Before starting this step, it is important to complete the notion of activation function introduced in the section 2.2.3.
It is often difficult to choose the ”ideal” activation function for a layer of neurons. Nevertheless, it is an important choice when creating a model. As a reminder, the activation function defines, with respect to the value calculated by the neuron, if it should be activated or not (pre-activation value). The calculation of this value can be seen on Figure 2.6 and is denoted A_j^1 the activation function is denoted Z. Figure 5.1 shows the graphical representation of famous activation functions (Step binary, Sigmoid, Relu, Softmax). There are others like Tanh or Maxout.

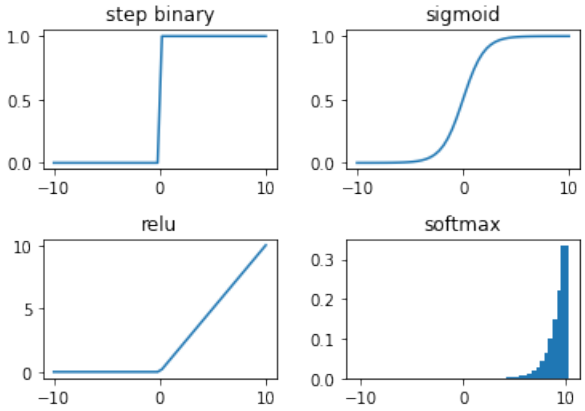

Figure 5.1: Representation of activation functions

To better understand how to choose the right activation function, this section looks at the two most popular: Relu and Softmax. First of all, Relu is often **used for hidden layers**. it allows that if the pre-activation value is less than zero, then the value is 0. In contrast, if the pre-activation value is bigger than zero, the function returns a value. Also, Figure 5.1 shows that after 0, Relu is linear. Relu is often used in hidden layers because, due to its linearity after zero, the gradient is linear too. In the article *ImageNet Classification with Deep Convolutional*, it is said that "Relu is 7 times faster than the Tanh or Sigmoide activation function". Relu allows the non-saturation of its gradient[4], which greatly accelerates the convergence of the stochastic gradient descent [23]. Then, Softmax activation function is **used for output layer**. It allows to create a "distribution function", i.e. the cumulative probability that an image belongs to a category does not exceed one. Softmax is not only have as input the pre-activation value of a neuron but of all the neurons of output layer. Moreover, this probability distribution allows to see how sure the model is of the classification it makes. Finally, in view of the advantages generated by these two functions, it is advisable to use them when creating a model. Nevertheless, the use of other activation functions is not excluded but requires prior research to understand their contribution to the model.

Model From scratch

To create a model from scratch, it is necessary to have some knowledge of ML. It is therefore important to look at famous network architectures to understand their specificities. Indeed, during creating a model from scratch, it is always interesting to look at models that have been approved by the scientific community. In order to create this from scratch model, the structure of the AlexNet model has been analyzed.

Figure 5.2 shows the structure of AlexNet [23]. The main layers used are convolutions and max pooling. On the one hand, the convolution layers have different parameters. These were introduced in section 2.4.1 and shows that it is sometimes interesting to change strides and/or padding. AlexNet uses Relu as an activation function after each convolution layer which allows to add non-linearity. Moreover, AlexNet ends with two full connected layers which also have Relu as activation function [23]. These two full connected layers allow to proceed to the prediction of the categories based on the extraction of features realized in the previous layers.

[4]Non saturation of the gradient implies that despite many iterations, the weights can still be updated. Indeed, in a case of saturation with a Sigmoid for example. When the value has left the linear part of the Sigmoid (its center) it is at an extremity that acts as an asymptote. This means that a change in the value of the gradient has little effect. Therefore, in order to change the value of the gradient, it takes many iterations

Therefore, it is these two layers that allow to perform the classification directly in the model as shown in Figure 2.9. On the other hand, the pooling layers allow to reduce the dimensions and consequently the computation time of the model. The output layer of AlexNet is a layer containing 1000 neurons (1000 categories) and takes as activation function Softmax.

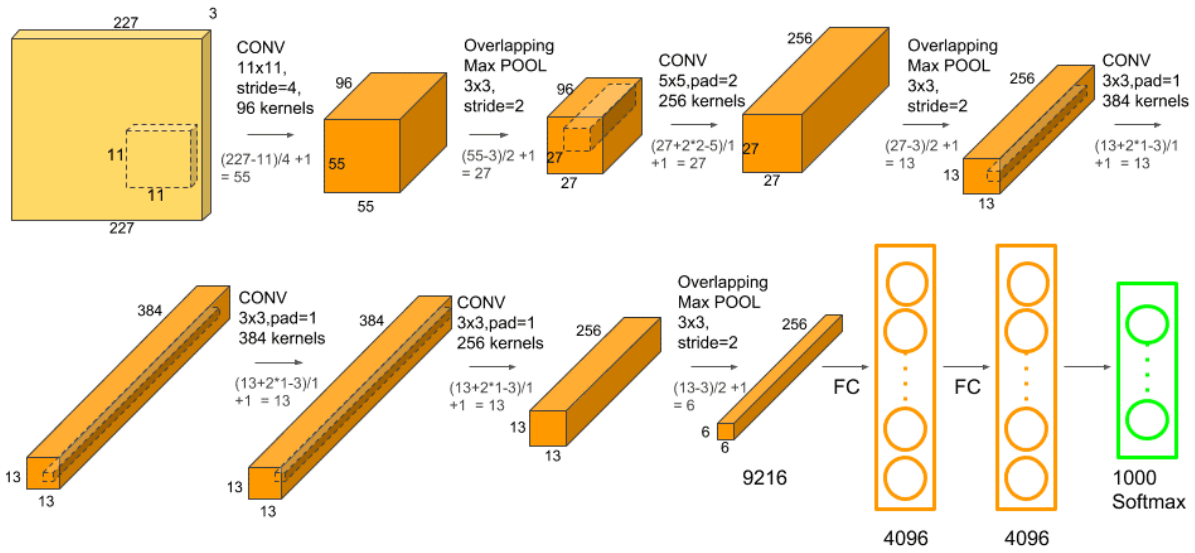

Figure 5.2: AlexNet architecture [41]

To conclude, creating a model from scratch requires a good mathematical understanding and knowledge of CNN. It is therefore a good practice to be inspired by a model validated by the scientific community. Moreover, when adding layers, it is necessary to pay attention to several things. On the one hand, the input layer needs to have information about dimension of the image and the output layer must have as many neurons as categories to be estimated. It is also important to underline that it is possible to create one's own layers with Keras and Tensorflow[5]. The creation of a model from scratch for MNIST dataset is already available on the Keras website but can be found in the appendix .4.

Model from transfer learning

The other solution for the creation of a model is to use a library (Tensorflow or Keras) to download a model that has already be trained on ImageNet. This method allows the recovery of the already trained model, which has several advantages: the non-necessity to train the model (saving time and computational costs) and the use of an architecture that has already been approved by the scientific community. It is difficult to say which model is the best for a given situation. Nevertheless, this framework aims to create a model that is stored in a device. Therefore, the most interesting models are small models designed for mobiles like: Mobilenet, MobilenetV2, NastNetMobile. In order to adapt the model to situation, the model is downloaded in its entirety except for the output layer in order to add only the one that suits in terms of the number of categories and the activation function. It is also possible to add layers at the end of the model.

[5] https://keras.io/guides/making_new_layers_and_models_via_subclassing/

Code 5.4 shows the downloading of a model and the addition of 3 layers. Moreover, the script contains 3 variables to create the model. The first one is the size of the input images. The second variable is the activation function for output layer and the last variable is the number of categories. It is important to note that the basis of this model is InceptionV3. To change the model, it is necessary to change all the inceptionV3's to the name of the model that transfers inside the script.

The other models are available on Keras[6]. For this step, the script is not a function because downloading another model requires another import. To create a function, it would require to import all the models provided by keras.

Source Code 5.4: Script to create model from Transfer Learning (InceptionV3)

```
# change with the name of the model that you want to use
from keras.applications import InceptionV3
from keras.models import Model
from keras.layers import Dense
from keras.layers import Flatten
# Three variables
input_size= "your image dimension" #(tuple)
output_activation_function ="name_activation_function"
num_classes = len(train_generator.class_indices)
# load model without classifier layers
model_from_transfer_learning =
 ↪  InceptionV3(include_top=False,input_shape=input_size)  #
 ↪  change the input if it's need
# add new  layers
layer1 = Flatten()(model.layers[-1].output)
layer2 = Dense(1024, activation='Relu')(layer1)
output = Dense(num_classes,
 ↪  activation=output_activation_function)(layer2)  # choose
 ↪  your own activation function for output layer
model_from_transfer_learning = Model(inputs=model.inputs,
 ↪  outputs=output)
```

5.2.5 Model compilation

The step after the creation of the model is the compilation of the model. For this step, the model created in the previous step is retrieved and meta-parameters are assigned to it: the optimizer and objective function. For this step a function has been created and can be found in Code 5.5. Firstly, The first three parameters of this function are the previously created model, the training data and the test data. Secondly, the next parameter is the optimizer. This one is the algorithm that changes the parameters of the model to reduce the error (the loss) as much as possible. Keras and Tensorflow propose several optimizers[7]. Performing an exhaustive analysis of all the optimizers and their performance is out of the scope of this thesis. Therefore, this section focuses on two popular optimizers: SGD and Adam. On the one hand, SGD stands for stochastic

[6] https://keras.io/api/applications/

[7] https://www.tensorflow.org/api_docs/python/tf/keras/optimizers

gradient descent. This optimizer is very common and has been explained in the section 2.2.3. On the other hand, the Adam optimizer is more and more used. The mathematical aspect of this optimizer is not explained in this thesis because it requires extensive knowledge in this field. However, Adam's algorithm can be found at appendix .5. Indeed, the explanation of the SGD at section 2.2.3 provides insight into the intuition behind CNN optimizers. The article : *ADAM : a method for stachastic optimization* tells that "Adam is designed to combine the advantages of two recently popular methods: AdaGrad (Duchi et al., 2011), which works well with sparse gradients, and RMSProp (Tieleman Hinton, 2012), which works well in on-line and non-stationary settings"[21]. The sparse gradient is a situation where the network fails to perceive strong enough signals for the model to adjust its weights well [9] and the online settings means that the model can be incremented with new data [26]. It is difficult to choose an optimizer and at the moment there are no real criteria to decide which optimizer is the best. Nevertheless, the scientific literature seems to agree that Adam has many qualities and allows a rather fast learning. The article quoted above says that Adam are "that the magnitudes of parameter updates are invariant to rescaling of the gradient, its stepsizes are approximately bounded by the stepsize hyperparameter, it does not require a stationary objective, it works with sparse gradients, and it naturally performs a form of step size annealing" [21] . Moreover, it is also said that Adam is very useful for "large datasets and/or high-dimensional parameter spaces". Therefore, the choice of a first optimizer may turn to Adam. Thirdly, the fifth parameter is a loss function. This one is the function that optimizer tries to minimize. In Keras and Tensorflow, there are three types of loss functions: probabilistic losses, regression losses and Hinge losses for "maximum-margin" classification. In this thesis, regression losses is only discusses but all the information about these different loss functions can be found on the Keras[8]. There are two often used loss functions: MSE and categorical cross entropy. On the one hand, mse (mean square error) is introduced in the section 2.1. It allows to calculate the difference (the error) between the estimated category and the true category. On the other hand, categorical cross entropy creates a list containing the probability that an element belongs to each class. For example, for the MNIST dataset, cross entropy calculates for each input data (digit image), the probability that it belongs to each digit. It is difficult to choose which loss function to take. Nevertheless, the paper *Cross-Entropy vs. Squared Error Training: a Theoretical and Experimental Comparison* suggests that it is better to use cross entropy for classification because MSE does not "punish" misclassified items enough. Nevertheless, MSE for regression may be a good choice because it can better capture the difference between two "close" data [15]. The last paramaters is the numbers of epochs. This number defines the number of times the model must go over the dataset. There is no perfect number of epochs, it depends on the validation and training error. Indeed, from the moment when the training error decreases but the validation error increases, we can suppose that the model undergoes overfitting (this term is explained in the following section). Therefore, it is judicious to stop at the number of epochs before this point.

To continue the construction of the two models, the appendix .6 is the script to compile the model created from scratch with the MNIST dataset. Code 5.5 contains the script to compile the model with transfer learning (InceptionV3) and data stored in drive.

[8]website (see `https://keras.io/api/losses/`)

Source Code 5.5: Function to compile model from Transfer Learning (InceptionV3)

```
def compile_and_fit_model_from_transfer_learning(model,
 ↪  train_generator, test_generator,
 ↪  optimizer='Adam',loss_function='categorical_crossentropy',
 ↪  nb_epochs=5):
    """
    compile and fit the model

    :param: the model that you want to compile and fit
 ↪  (tensorflow.python.keras.engine.sequential.Sequential)
    :param train_generator : the train generator create in
 ↪  the before step with the function load_your_data()
 ↪  (directoryIteror)
    :param: test_generator : the test generator create in
 ↪  the before step with the function load_your_data()
 ↪  (directoryIteror)
    :param optimizer : optimizer is the algorithm to
 ↪  calculate the gradient by default it's 'Adam' (string)
    :loss_function : it is the loss function to calculate
 ↪  the error (loss) by default it's
 ↪  'categorical_crossentropy' a famous loss function is
 ↪  mean square error (str)
    :nb_epochs: number of times the model runs over all data
 ↪  (int)

    return the model fitted.
    """
    model.compile(optimizer=optimizer, loss=loss_function,
     ↪  metrics=['accuracy'])  # compilation du modèle
    step_size_train = train_generator.n //
     ↪  train_generator.batch_size
    # Total number of steps (batches of samples) before
     ↪  declaring one epoch finished and starting the next
     ↪  epoch
    number_samples_validation = test_generator.n
    number_validation_steps = number_samples_validation //
     ↪  test_generator.batch_size
    step_size_train = train_generator.n //
     ↪  train_generator.batch_size
    return model.fit(train_generator,
     ↪  steps_per_epoch=step_size_train, epochs=nb_epochs,
     ↪  validation_data=test_generator,
     ↪  validation_steps=number_validation_steps)
model_fitted =
 ↪  compile_and_fit_model_from_transfer_learning('your_model',
 ↪  'your_train_generator', 'your_test_generator')
```

5.2.6 Model Evaluation

A crucial step in the creation of a CNN is its evaluation. There are several processes to evaluate its relevance: accuracy rate of the test dataset, confusion matrix, display of the probabilities of an image belonging to a category and saliency maps. Before discussing these four procedures, it is important to introduce the notions of overfitting and underfitting. On the one hand, overfitting refers to a model that has learned the data "stupidly", i.e. it has learned the data by heart but has not learned any rules to extract features from the model. In a case of overfitting, the accuracy rate for the training data is very high but the accuracy level for the test data is low. Figure 5.3 shows via the blue line how the model has learned all the data by heart. On the other hand, underfitting is a model that has not learned to classify the data well. The most common reason for underfitting is a lack of training data or a model that is too simple to perform complex classification. Underfitting is easily noticeable because it has a low accuracy rate for all three databases (training, test, validation).

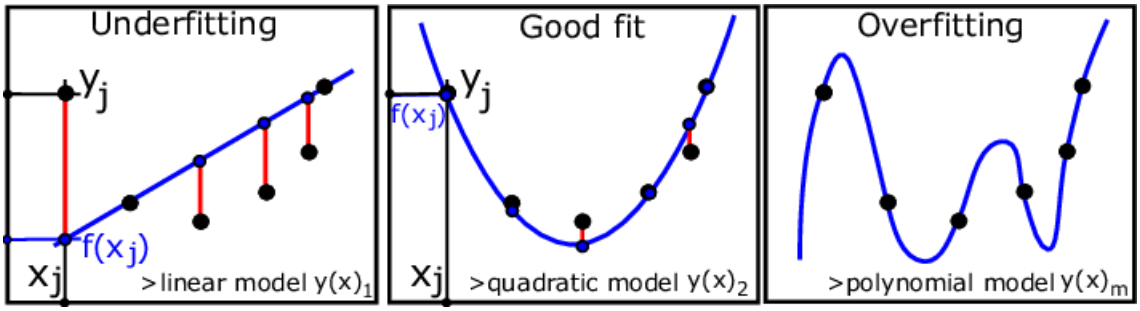

Figure 5.3: Representation underfitting, good fit and overfitting [37]

The first way to evaluate a model is to look at the accuracy rate of the validation dataset provided when calling the "compile_model" function. Moreover, thanks to the accuracy_test_dataset function call, it is possible to see the accuracy of the model on the training data. The function takes as parameters a model and a path to access the test data and it can be found in Code 5.5.

The second process is the confusion matrix, this matrix allows to see for each category if it is well classified or not and to see with which category the model is wrong. The perfect normalized confusion matrix is composed only of 1's in the diagonal and 0's everywhere else as shown on Figure 5.4. This means that the model does not confuse any category.

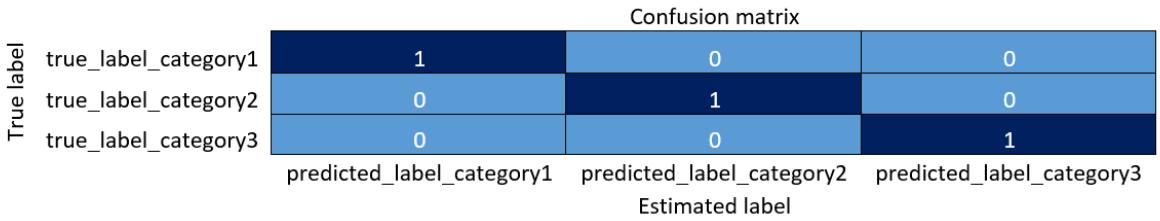

Figure 5.4: Representation of the perfect normalized confusion matrix

This matrix is, useful to quickly visualize the error of the model and to spot its weakness. For example, for the MNIST dataset, there could be a lot of confusions between the 3 and the 5 because these numbers are similar in the way they are written. Code 5.6 is a function that returns the normalized confusion matrix (all values between 0 and 1) and takes as parameter the link to access the test data and the model.

Source Code 5.6: Function to create matrix confusion

```
def confusion_matrix(model, path_for_test_generator):
    """
    Show the confusion matrix
    Parameters
    :param: model that you want the confusion matrix (class
    ↪ functional)
    :param: path where the test generator is stored
    ↪ (directory_iterator)
    """
    import pandas as pd
    import seaborn as sn
    import matplotlib.pyplot as plt
    from keras.preprocessing import image
    from keras.applications.mobilenet import
     ↪ preprocess_input
    from keras.preprocessing.image import ImageDataGenerator
    test_datagen =
     ↪ ImageDataGenerator(preprocessing_function=preprocess_input)
    test_generator =
     ↪ test_datagen.flow_from_directory(path_for_test_generator,
     ↪ target_size=(224, 224), color_mode='rgb',
     ↪ batch_size=32,
    class_mode='categorical', shuffle=False)

    probabilities = model.predict(test_generator)
    probabilities_norm = probabilities / probabilities.max()
    y_pred = np.argmax(probabilities, axis=1)
    data_norm = {'y_Actual': test_generator.labels,
                 'y_Predicted': y_pred
                 }
    df = pd.DataFrame(data_norm, columns=['y_Actual',
     ↪ 'y_Predicted'])
    confusion_matrix = pd.crosstab(df['y_Actual'],
     ↪ df['y_Predicted'], rownames=['Actual'],
     ↪ colnames=['Predicted'])
    class_indices = test_generator.class_indices
    sum = confusion_matrix.sum(axis=1)
    sum = sum.to_list()
    for i in sum:
```

```
        confusion_matrix[sum.index(i)] =
         ↪  confusion_matrix[sum.index(i)] / i
        for key, value in class_indices.items():
            if sum.index(i) == value:
                confusion_matrix = confusion_matrix.rename(
                    columns={sum.index(i): key},
                    index={sum.index(i): key})
    sn.heatmap(confusion_matrix, annot=True)
    plt.show()

confusion_matrix('your_model', 'your_path_to_data')
```

The third process is to choose images from the test dataset to see how confident the model is in its classification. This process provides less information than the first two but allows to see how sure the model is of classifying images. Code 5.7 is a function that returns the probability of an image to belong in each category. This function offers two possibilities: to see the probability for a random image or to see the probability for a specific image (use the path_specific_image parameter). As a reminder, if the activation function of the last layer is a Softmax, the sum of the probability of each classification gives 1.

Source Code 5.7: Function to get the probability that image belongs to a class

```
def probability_for_image(model,
 ↪  path_for_test_generator,path_specific_image=" "):
  """
  Displays the probability that the image belongs to each
 ↪  class
  Paramèters
  :param: model that you want the confusion matrix (class
 ↪  functional)
  :param: path where the test generator is stored (string)
  :param: path if you want to test for a specific picture
 ↪  (string)
  """
  from keras.preprocessing import image
  from keras.applications.mobilenet import preprocess_input
  from keras.applications.mobilenet import preprocess_input
  from keras.preprocessing.image import ImageDataGenerator
  import os
  import random

  test_datagen =
   ↪  ImageDataGenerator(preprocessing_function=preprocess_input)
  test_generator =
   ↪  test_datagen.flow_from_directory(path_for_test_generator,
  target_size=(224, 224),color_mode='rgb',
  batch_size=32,class_mode='categorical',shuffle=False)
  if path_specific_image == " ":
```

```
    dico_path ={}
    for classes in os.listdir(path_for_test_generator):
      for img in
       ↪  os.listdir(path_for_test_generator+'/'+classes) :
        dico_path[img]=
         ↪  path_for_test_generator+'/'+classes+'/'+img

    name_image = random.choice(list(dico_path.keys()))
    #img_path.split()
    print('Image name is :',name_image.split('.')[0])
    #print('The path of the images is
     ↪  ',dico_path.get(img_path))
    img = image.load_img(dico_path[name_image],
     ↪  target_size=(224, 224))
  else :
      print('Image path is :',path_specific_image)

      img = image.load_img(path_specific_image,
       ↪  target_size=(224, 224))

  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)
  x = preprocess_input(x)
  num_classes= test_generator.class_indices
  inv_map = {v: k for k, v in num_classes.items()} #allow to
   ↪  return the dict key become value
  score = model.predict(x)
  score= score[0].tolist()
  for i in score:
    pourcent= i*100
    print('This image is %.2f
     ↪  %%'%(i*100),inv_map[score.index(i)])

probability_for_image('your_model',
 ↪  'your_test_generator_path')
```

The last process is the saliency maps which aim to show the pixels that are important for the classification. Saliency maps show the importance of each pixel for the classification of a image. This allows to see where the model is going to look for information to classify and to see if the image focuses on the right pixel areas to sort.

Figure 5.5 below shows an example of a saliency map that clearly shows which area the model is basing the classification on.

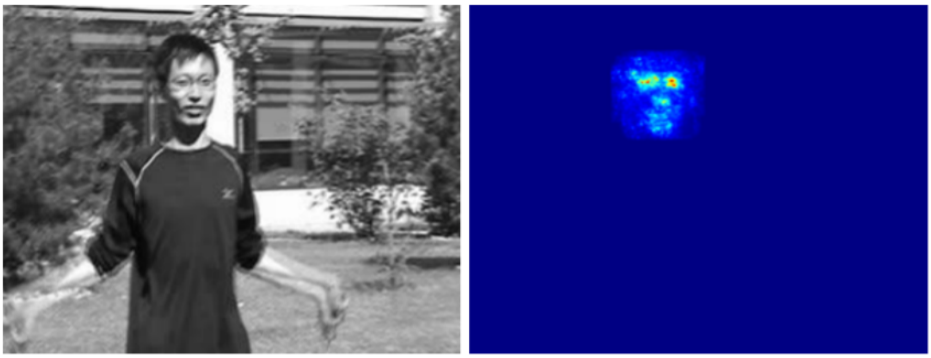

Figure 5.5: Representation of saliency maps [18]

Code 5.8 shows a function that returns a saliency map. It takes as parameters: the model, the path to the test_generator and the path to the image where the saliency map should be applied.

Source Code 5.8: Function to get saliency map

```
def saliency_map(model, path_for_test_generator,
 ↪  image_path):
    """
    Print the saliency map for the image
    Parameters
    :param model: the model with which you want to see the
 ↪  saliency map (model)
    :param path_for_test_generator : path where test date
 ↪  are stored (string)
    :param image_path : path where the image are stored
 ↪  (string)

    return print the saliency map
    To work, you must use this function after compiling the
 ↪  model
    code modified from :
    https://usmanr149.github.io/urmlblog/cnn/2020/05/01/
    Salincy-Maps.html
    """
    import numpy as np
    import tensorflow as tf
    from keras.applications.mobilenet import
     ↪  preprocess_input
    from keras.preprocessing.image import ImageDataGenerator
    test_datagen =
     ↪  ImageDataGenerator(preprocessing_function=preprocess_input)
```

```
    test_generator =
     ↪  test_datagen.flow_from_directory(path_for_test_generator,
     ↪  target_size=(224, 224), color_mode='rgb',
     ↪  batch_size=32, class_mode='categorical',
     ↪  shuffle=False)

    _img = keras.preprocessing.image.load_img(image_path,
     ↪  target_size=(224, 224))

    # preprocess image to get it into the right format for
     ↪  the model
    img = keras.preprocessing.image.img_to_array(_img)
    img = img.reshape((1, *img.shape))
    y_pred = model.predict(img)

    images = tf.Variable(img, dtype=float)

    with tf.GradientTape() as tape:
        pred = model(images, training=False)
        class_idxs_sorted =
         ↪  np.argsort(pred.numpy().flatten())[::-1]
        loss = pred[0][class_idxs_sorted[0]]

        grads = tape.gradient(loss, images)
        dgrad_abs = tf.math.abs(grads)
        dgrad_max_ = np.max(dgrad_abs, axis=3)[0]
        ## normalize to range between 0 and 1
        arr_min, arr_max = np.min(dgrad_max_),
         ↪  np.max(dgrad_max_)
        grad_eval = (dgrad_max_ - arr_min) / (arr_max -
         ↪  arr_min + 1e-18)
        fig, axes = plt.subplots(1, 2, figsize=(14, 5))
        axes[0].imshow(_img)
        i = axes[1].imshow(grad_eval, cmap="jet", alpha=0.8)
    return fig.colorbar(i)


saliency_map('your_model', 'path_test_data',
 ↪  'path_your_image')
```

To conclude, this processes to evaluate the model are complementary and they provide a good overview of the quality of produced model.

5.2.7 Model saving and loading

This second to last step is a facilitation step. Indeed, it is not mandatory for the creation of a model. However, it is useful to be able to save the model. Code 5.9 shows a function to save model. This function takes as paramaters the location where the model will be stored.

Source Code 5.9: Function to save model

```
def saving_model(model,path):
  """
  save the model
  parameters
  :param model: model that you want to save (model)
  :param path: location where you want to save your model
 ↪  (string)
  NB path have to finish with the extension h5
  exemple :path/yourmodel.h5
  """
  import Tensorflow as tf
  import keras
  tf.keras.models.save_model(model,path)
  print('Your model is saved')

saving_model('your_model','your_path')
```

It is also necessary to be able to load a model that has been made previously. Code 5.10 shows a figure to load previously created models. This function takes as parameters the path where the model is stored.

Source Code 5.10: Function to load model

```
def loading_model(path):
  """
  Load the model from the path
  Parameters
  :param: path where the model is stored
  return the model that you have been downlaoded
  """
  import keras
  import tensorflow as tf
  from keras.models import Model
  return tf.keras.models.load_model(path)

model =loading_model('path_to_model')
```

5.2.8 Model conversion

The purpose of this last step is to transform model from Tensorflow to Tflite. The Tensorflow models are made to be used on a computer while tflite models are made to be used on a device. This conversion allows to reduce the size of the model while keeping an almost similar level of accuracy. The flutter documentation does not explain the changes made during the conversion to reduce the model. The function to perform this operation can be found in Code 5.11 below.

Source Code 5.11: Function to convert model in Tflite model (model for device)

```
def convert_tflite(loading_path, saving_path):
    """
    Convert the model to a tflite model
    Paramaters
    :param loading_path : location where the model in stored
    in h5 extension
    param saving_path : location where the model is saved in
    tflite extension
    """
    import tensorflow as tf
    model = tf.keras.models.load_model(loading_path)
    converter =
     tf.lite.TFLiteConverter.from_keras_model(model)
    converter.experimental_new_converter = True
    tflite_model = converter.convert()
    open(saving_path, "wb").write(tflite_model)
    print("model is converted!")

convert_tflite("your_loading_path", "your_saving_path")
```

5.3 Conclusion

This section talks about steps to create a CNN. The different steps have been shown by code examples in two cases. On the one hand, the case where there are data stored in drive and a model is built from transfer learning. On the other hand, case where there are data from library and model is built from scratch. Nevertheless, there are four possible scenarios and only two have been discussed as shown in the table below. The choice to show 2 models (2 scenarios) is made for the reason of clarity. However, the script to create a model from scratch and with data stored in drive is available in appendix .7.

Type of model	Type of data	Codes
From scratch	Dataset from tf	during section 5.2
From transfer learning	Own data	during section 5.2
From scratch	Own data	Appendix .7
From transfer learning	Dataset from tf	Not useful

Table 5.1: Scenarii possibles to create a model

To keep in mind from this chapter

1. Choosing library
   (a) Tensorflow
      ☞ Deployment service
      ☞ Large community
      ☞ Better literature
   (b) Pytorch
      ☞ Dynamic graph
2. Application structure
   (a) Libraries importations
   (b) Data downloading
   (c) Data exploration
   (d) Model creation
   (e) Model compilation
   (f) Model Evaluation
      ☞ Accuracy rate of test dataset
      ☞ Confusion matrix
      ☞ Display of the probabilities of an image belonging to a category
      ☞ Saliency maps
   (g) Model saving and downloading
   (h) Model conversion tflite

Chapter 6

Application creation

In contrast to a machine learning model (CNN), it is complicated to establish precise steps to create an application. Indeed, for the creation of an application, this is more complicated because it is subject to many personal factors such as: application color, button size, button colors, how to store the model and so on. In view of this difference, this chapter focuses on the one hand on the explanation of the application set up for this project through fourth sections which are: choice of the technology used, introduction to Flutter, structure of the application and code explanation. On the other hand, the main steps for the creation of the application. the files for the creation of the application are from the appendix .8 to the appendix .10.

6.1 Choosing technologie for application

There are several languages and frameworks to create a mobile application. For the realization of this project, A framework is used that is higher level and because it allows to reach a functional application more quickly. In addition, the use of frameworks allows to have cross platform. This means that the same code works for Android devices and IOS devices. The two biggest frameworks currently used to create a mobile application are Flutter and React Native. It was difficult to picking one but the choice was made for Flutter which has a better literature. Moreover, Flutter is a framework designed by Google like Tensorflow. It is easier to choose technologies created by the same company because it allows to link them easily. Another possibility is to use a native language like java or swift. However, this possibility requires a better knowledge in computer science and does not allow cross-platform.

6.2 Flutter introduction

Flutter was created in 2017 and is based on c and c++ as a language. The biggest advantage of Flutter is that with a single code, the application can be used on Android and IOS. The programming language is Dart which is also created by Google [6].
In order to make the framework as simple as possible, Flutter has implemented widgets. These are the center of the framework. Indeed, widgets can be seen as objects as in OO (oriented-object) that have already been created beforehand by Flutter. Moreover, these widgets created by Google ensure an optimal rendering that adapts to the device. Therefore, the rendering of a widget on Android and IOS varies to look like the styles that the device provides. These widgets are nested together to create the application. There are different categories of widgets: those that define a position on the phone, those to interact directly with the user (button),those to store others and so on. An example of a widget to store other widgets is column() which allows to display several widgets one below the other. Figure 6.1 shows an example of a widget for Flutter's "Hello World". In this Figure, the first widget is MyApp and it contains all the other widgets in the application. A concrete example of the column widget is also visible. Indeed, this widget contains 2 texts that are displayed one below the other. There are two types of widget: stateless widget and stateful widget. Stateless widgets are the equivalent of immutable objects in Java. This means that their "state" remains unchanged during the whole execution. In contrast, stateful widgets are mutable and vary according to execution. It is also important to note that it is also possible to create personal widgets.

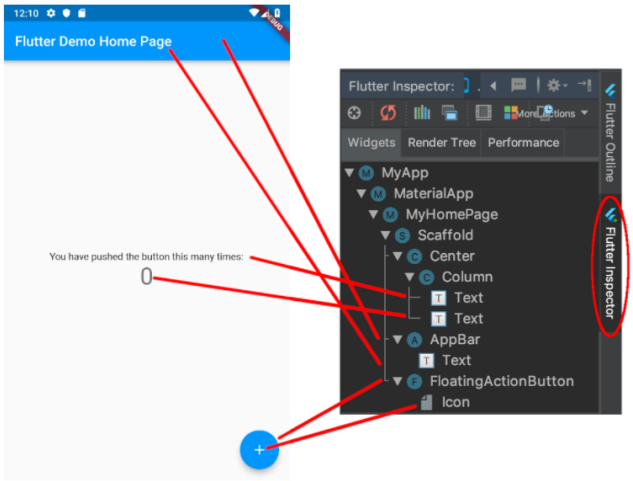

Figure 6.1: Overview of widgets operation [7]

6.3 Application structure

In this section, the architecture of the application is explained. First of all, the management of dependencies and packages to be downloaded for the application is in the pubspec.yaml file. For the application, tflite and image_pickers packages have been added. The tflite package allows Flutter to have access to functions to read tflite files (CNN) and the image_picker package is used to get images from its gallery/photos. The yaml file is used to store all the metadata, an example of this file can be found in the appendix .8 and a part of this files is below (see Code 6.1). Therefore, this file contains the version of application, the SDK (software development toolkit) used, the project description, etc. It is also in this file that the access path where the images and models are stored is specified.

Source Code 6.1: Part of the yaml file

```
version: 1.0.0+1

environment:
  sdk: ">=2.7.0 <3.0.0"

dependencies:
  flutter:
    sdk: flutter

  # The following adds the Cupertino Icons font to your
   ↪  application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^0.1.2
  tflite: ^1.0.4
  image_picker: ^0.6.2+4
[...]
```

Next, Code written in dart is divided into three large files: Main.dart, Home.dart and MyApp.dart. File Main.dart allows to call the general function RunApp() which launches the application. Moreover, it is also in this file that the paths to the other dart files are found. In general, a dart file represents a page of the application. Therefore, the Main file allows to call the function that launches the application and to instantiate the routes to the other dart files. Code 6.2 shows the Main file.

Source Code 6.2: Main.dart

```
import 'package:flutter/material.dart';
import 'package:name_application/pages/interface_1.dart';
import 'package:name_applications/pages/Interface_2.dart';

void main() => runApp(MaterialApp(
    debugShowCheckedModeBanner: false,
  initialRoute: '/path_1', " #initial path (first screen
   ↪  that you see)
  routes: {
    '/path_1':(context)=>Interface_1(),
    '/path_2:(context)=> Interface_2()

  }
));
```

The second dart file is Home.dart as its name indicates it is the file to create the menu interface. The last dart file MyApp is the file that contains the interface for the classification of the model and the different functions to get there.

In addition, for the model to run on the phone. It is necessary to add in the file build.gradle in the Android folder the Code 6.3 below. This will make the tflite package work.

Source Code 6.3: Build.gradle file (Android folder)

```
    aaptOptions{
        noCompress 'tflite'
        noCompress 'lite'
    }
```

Finally, an asset file with all the documents imported for the realization of the project has been created. This file is located at the same level in the tree structure as the files: build, gen or android.

6.4 Code explanation

This section explains the Home and MyApp files in more depth. The goal is to understand the code as well as possible. The explanation of the code aims to make it more easily reusable and modifiable.

Home file

Home file is used to create home page. To do this, it is necessary to create the ”Home” class which is a stateful widget. This class returns a Scaffold (Flutter widget) which contains the whole interface of the page. Before returning this widget, variables for the height and width are created. These variables are used to give relative positions to the widgets according to the dimension of the device. In the scaffold, the background color is defined and the body of the scaffold is a SingleChildScrollView widget. This widget is useful when the interface requires only one space and it is fully visible on the screen according to the Flutter doc. This last widget centers all the elements and contain another container widget (widget designed to nest widgets). The set of widgets of the container is not detailed but they can be classified in three parts. The first part allows to display the title and the background image. The second part is three containers that allow to create the three buttons: Image classification, Button 1 and Button 2. Each container represents a button, and it is in these containers that the type of button, the color, the logo, the access to another page when the user presses, are determined. The third part is a row widget which allows to display the UNamur logo. For this part, the Row widget has been chosen so that several images can be put one after the other. Code 6.4 below shows an example of how to start creating a menu interface. The entire Code for this interface can be found in the appendix .9.

Source Code 6.4: Home.dart

```
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
class Home extends StatefulWidget {
  @override
  _HomeState createState() => _HomeState();
}

class _HomeState extends State<Home> {
  @override
  Widget build(BuildContext context) {
    //Variable instanciation
    var device_Size=MediaQuery.of(context).size;
    var device_height= device_Size.height;
    var device_width= device_Size.width;
    return Scaffold(
      backgroundColor: Colors.white,
      // Beginning of the Widget SingleChildSctrollView
      body:SingleChildScrollView(
        child : Center(
          //Beginning of the Containers
          child:Container(
```

```
            child: Column(
              // First part
              children: <Widget>[
                Container(height: device_height* 1/2 /*250
                 ↪  */ ,
                    decoration: BoxDecoration(
                        image: DecorationImage(
                            image:
                             ↪  AssetImage('assets/background.png'),
                            fit: BoxFit.fill
                        )
                    )
                            [......]
```

MyApp file

MyApp() file is used to create the interface for the object classification. Like the Home class, the MyApp class is a mutable class. This class contains three attributes: the file where the model is stored, the output list and a boolean indicating if the image is loaded. This class is divided into three main parts: the class state function, the interface function and the asynchronous functions (working functions).

First of all, the state function allows to change the values of the attributes and to ”notify” the class if the model has been loaded or not. This function does not return anything, it is just used to indicate the execution state of the class. Then, there are four functions to create the interface. The build() function allows to create the Scaffold which returns the whole interface (Widget object). This one is an AppBar which is a widget that defines a default display (see Figure 7.7). This AppBar allows to display the button which opens a dialog box when it is pressed. This function contains the 3 other functions that have a role for the display. Function show_choicedialog() creates box that allows to ask the user if he wants to take a picture or to choose a picture in these files. When the user has chosen the picture, it appears with the predicted label and the accuracy of the prediction. To display this, the build function calls the function output_text(). The last function (decide_image_view()) allows to create the message before selection of an image by the user. Finally, this class contains 5 asynchronous functions that allows the interface to work. The first function is Load_model() as its name indicates, it allows to download model and lables of categories from assets folder. There is also a dispose function which allows to close the file after opening it. The second function Classify_image() takes as parameter the image and uses the model downloaded in the previous asynchronous function. This function also changes the status of the attributes by determining an output and changing the value of the loading boolean. The third function pick_image allows to retrieve images from a gallery, it also changes the value of the image and loading attributes. Moreover, it calls the Classify_image() function to classify the image. The fourth function (open_gallery()) and the last function (open_camera()) are asynchronous functions that create access to gallery and camera. Both change the attributes of the MyApp function to signal that an image has been taken and set the value of the image to be filed. Code 6.5 below shows an example of how to start creating a classification interface. The entire Code for this interface can be found in the appendix .10. Code for MyApp can be found in the appendix .10.

Source Code 6.5: MyApp.dart

```
import 'package:image_picker/image_picker.dart';
import 'package:tflite/tflite.dart';
import 'dart:io';
import 'package:flutter/material.dart';
class MyApp extends StatefulWidget {
  @override
  _MyAppState createState() => _MyAppState();
  }
// First part : State functions and
class _MyAppState extends State<MyApp> {
  // attribute initialization
  List _outputs;
  File _image;
  bool _loading = false;

  @override
  void initState() {
    super.initState();
    _loading = true;

    load_model().then((value) {
      setState(() {
        _loading = false;
      });
    });
  }

// Second part : creation interface function
  @override
  Widget build(BuildContext context) {
    return Scaffold(
        appBar:
      AppBar(
        title: const Text('Select a picture',textAlign:
         ↪  TextAlign.start),backgroundColor:
         ↪  Color.fromRGBO(97, 103, 215,0.8),
      ),
      body: _loading
          ? Container(
        alignment: Alignment.center,
        child: CircularProgressIndicator(),
    )
          : Container(
        width: MediaQuery.of(context).size.width,
        child: Column(
```

```
          crossAxisAlignment: CrossAxisAlignment.center,
          mainAxisAlignment: MainAxisAlignment.center,
          children: [

             decide_image_view(),
            SizedBox(
              height: 50,
            ),
            _outputs != null
                ?
            Text(
              output_text(),
              textAlign: TextAlign.center,
              style: TextStyle(
                color: Colors.black,
                fontSize: 20.0,
              ),


            )
                : Container()
          ],
        ),
      ),
      floatingActionButton:
       ↪  FloatingActionButton(backgroundColor:
       ↪  Color.fromRGBO(97, 103, 215,0.8),
        onPressed: (){
        show_choice_dialog(context);
        },
        child: Icon(Icons.image,
      ),
    ));
  }

                              [......]
```

6.5 Step to create application

After the explanation of application provided in this thesis. It is interesting to abstract the main steps to create a mobile application using deep learning. So, this section covers all to create application with Flutter framework.

6.5.1 Installation

There are 3 technologies to install to create the application. The two first technologies are Android studio and Flutter. This thesis does not explain the different steps for installing Flutter and Android studio. Indeed, there are several tutorials that do. Moreover, a video is more interactive than a series of handwritten explanations for the installation. Nevertheless, before to proceed to the other steps, it's necessary to be able to call this command: "flutter doctor" and have the result as in Figure below. This shows that all conditions are met to make Flutter work.

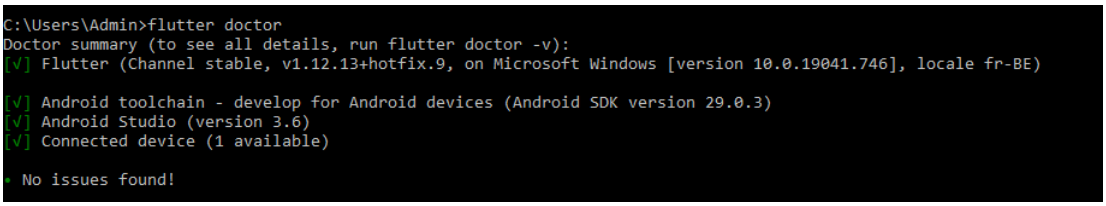

Figure 6.2: Console flutter

After receiving this output from the command prompt, it's possibility to download the Flutter plugin for Android studio. This step is done when launching the Android studio application before launching a new project.
The last technology is useful to test the application. In fact, it is necessary to use an emulator that allows to simulate a device on a PC or use a real device to see directly the interface of what is created. In addition, Flutter allows "hot reloads", which means that it is possible to see directly the changes made to the code on the device without having to compile and reinstall the application each time.

6.5.2 Adding depedencies

As introduced in the next section, any package needed for the application must first be imported via this file (pubspec.yaml). In addition, this files allow to specify the version of packages. The pubspec.yaml of the project can be found in appendix .8.

6.5.3 Dart script creation

This step is the real implementation step of this project. A dart file for each interface. Then, the Main file (created automatically when the application is launched) must be used to create the different paths to these files. When creating these files, several widgets are nested in order to keep the structure in mind, the flutter inspector feature proposed by Android studio allows to see the widgets tree. Finally, for each created function or widget, it is important to test it on the device. Indeed, this allows to see instantly the rendering of the application and to realize directly if there is a problem in the compilation.

6.6 Conclusion

It is complicated to create precise guidelines for creating an application. Nevertheless, this chapter explains the code of an application that can be reused and gives the main steps for creating application.

To keep in mind from this chapter

1. Choosing technology
   - ☞ Framework
     - Flutter : same creator as Tensorflow, bigger community, better literature
     - React native : older technology
   - ☞ Native programming Language: do not allow cross platform
2. Flutter introduction
   - ☞ All is about widgets
3. Application structure
   - ☞ Pubspec.yaml: management of dependencies and packages
   - ☞ Main.dart: call the general function RunApp() and instantiate the routes to the other dart files
   - ☞ Home.dart: creation menu interface
   - ☞ MyApp.dart: creation of the interface for classification
4. Step to create application
   - ☞ Installation of flutter, IDE, device simulator
   - ☞ Adding dependencies in pubspec.yaml
   - ☞ Creation of dart files

Chapter 7

Case study

The end of this thesis concludes with the application of all the theory covered and the scripts provided in the appendices. This part is divided into three main sections: the presentation of the case study, the explanation of the construction of the model and the different mock-ups of the application.

7.1 Case study presentation

One of the outcomes of this thesis is the creation of an application to help sorting waste. This application can be used on Android or on IOS. Nevertheless, this application is not deployed. Indeed, the objective of this thesis is to allow anyone who wants to create his application to be able to find all the theoretical information and the code to realize this project. For the creation of the CNN, the choice is made to use transfer learning with trashnet data that is downloaded on github[1] and using Google Collabs as a Notebook. Google collabs is a cloud service based on Jupyter[2] that allows to train models directly in the cloud. The use of this service allows to avoid the installation of resources on the PC [28].

In order not to duplicate the same work on the data, the trash category is not considered. The training dataset is therefore composed of 5 classes: cardboard (321 images), paper (466 images), metal (322), plastic (386) and glass (399). For the test dataset, it is composed of : 82 images of cardboard, 126 images of paper, 88 images of metal, 96 images of plastic and 100 of glass. These data are stored in a drive with the following file structure:

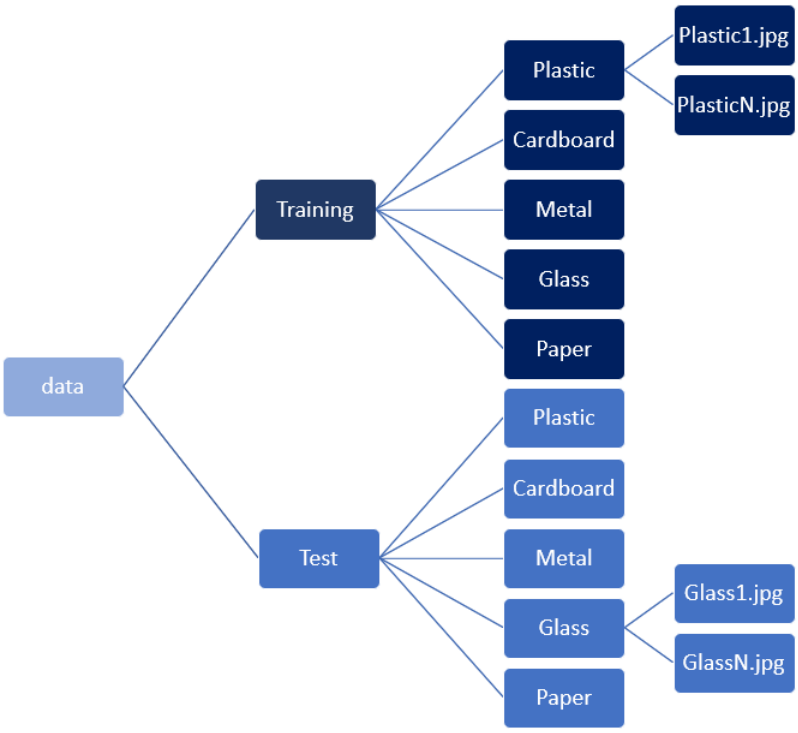

Figure 7.1: Overview of the file aborescence

For the creation part of the application, it is created thanks to Android studio. To realize this prototype application, the flutter framework is used. The application offers the possibility to take a picture of the element that user would like to classify or to go to the gallery to pick the picture. The choice also turned to the storage of the model within the device for reasons of data confidentiality.

[1] https://github.com/garythung/trashnet
[2] Jupyter is a online platform to make programmation

7.2 Building model

First, the library, data, and exploration download steps are not discussed in this section because they are essentially identical to the steps explained in sections 5.2.1, 5.2.2, and 5.2.3. Secondly, during creating the model several models is tested in order to find the model that gives the best performance. First, models chosen to perform the transfer learning are DenseNet121 , MobileNet, MobileNetV2 and NasNetMobile. These are chosen because they are small (and therefore easy to store on a device). For each model, the output layer is removed and three layers are added: a flat layer and two dense layers. These layers are added to show that it is possible to add layers to a model obtained from transfer learning and because they allow the inputs to be well resized to the output layer. The last layer is made of 5 neurons (5 categories) and is activated by the activation function Softmax. To complete the evaluation of the models, they are trained with and without data warping to see the impact on accuracy. In this case, data warping helps to improve accuracy of model. Moreover, models without data warping are tested with two optimizers: "SGD" and "Adam". Figure 7.2 shows the different accuracy rates achieved and the characteristics of these models. It has been decided to keep the DenseNet121 model with data augmentation and with the SGD optimizer. This one gives an accuracy for the test data of 90.04%.

Summary result					
Model name	Epochs	data_augmentation	optimizers	test accuracy	training accuracy
DensetNet121	5	no	SGD	86,99%	98,54%
MobileNet	5	no	SGD	85,36%	99,16%
MobileNetV2	5	no	SGD	74,39%	99,22%
NasNetMobile	5	no	SGD	77,03%	99,41%
DensetNet121	5	no	Adam	58,94%	69,86%
MobileNet	5	no	Adam	68,09%	75,05%
MobileNetV2	5	no	Adam	17,28%	86,69%
NasNetMobile	5	no	Adam	57,72%	96,11%
DensetNet121	5	yes	SGD	90,04%	99,81%
MobileNet	5	yes	SGD	85,77%	99,93%
MobileNetV2	5	yes	SGD	76,42%	99,70%
NasNetMobile	5	yes	SGD	67,68%	99,52%

Figure 7.2: Result of all trained models

Thirdly, after the selection of this model (DenseNet121), the different functions introduced in section 5.2.6 are used to ensure the performance of the DenseNet121 model.
The confusion matrix (see Figure 7.3) made it possible to realize that the most difficult category to classify for the model is the last class i.e. plastic. There are two reasons for this events. the first reason is that some plastics are very similar to glass. Indeed, the dataset contains plastic bottles where even a human being is confused between glass and plastic. The second reason is that plastic has no particular shape or color which makes it difficult to recognize.

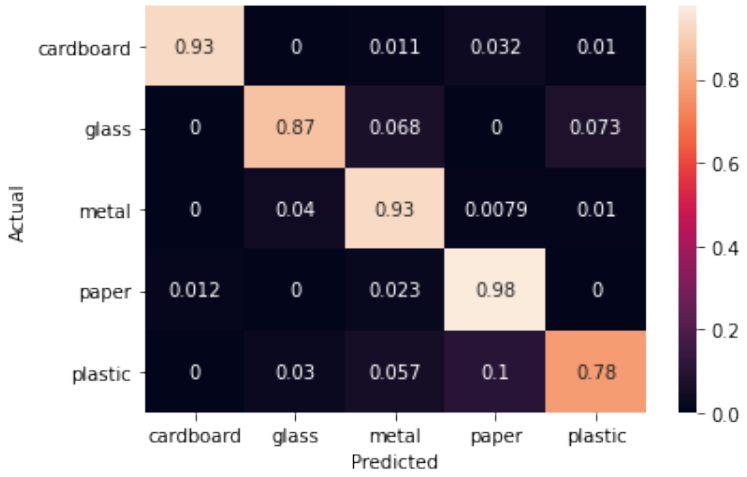

Figure 7.3: Confusion matrix for DenseNet121

Taking images to look at the certainty of classification showed good results in that the model is often certain of the classification it is making. Figure 7.4 below is an example of the return of the function. It is also interesting to use this function to understand which images are misclassified. In this case, this function was used to confirm the hypothesis that the model was confusing glass and plastic bottles.

```
Found 492 images belonging to 5 classes.
Image name is : glass493
This image is 0.00 % cardboard
This image is 99.88 % glass
This image is 0.00 % metal
This image is 0.00 % paper
This image is 0.12 % plastic
```

Figure 7.4: Exemple of probability_for_image() return

Saliency map allows to see which pixels are important for the classification of the model. This ensured that the model focused on the waste to be sorted and not its environment. Figure 7.5 below shows an example of a saliency map applied to the model.

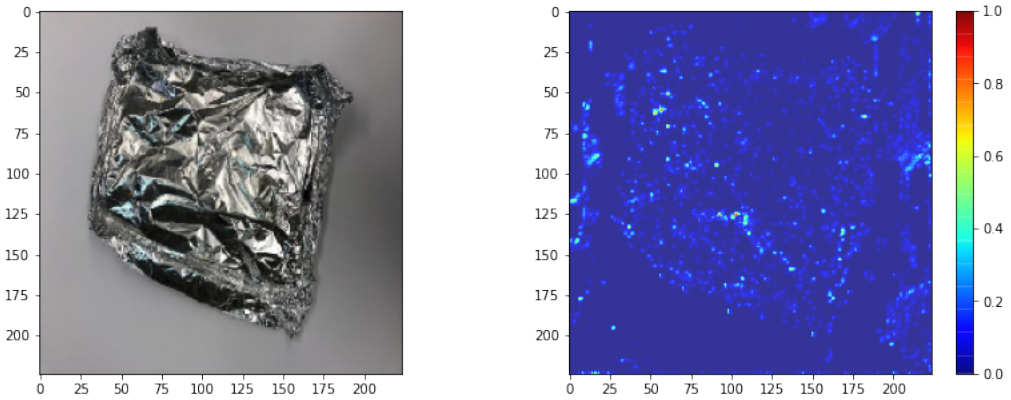

Figure 7.5: Exemple of saliency map for DenseNet121

Lately, the backup and transformation steps have also been performed but is not explained further for the same reasons as the first steps.

7.3 Application's mock-ups

As the part on the creation steps is introduced in the previous chapter, this section only talks about the different mock-ups of the application.
On the one hand, the home menu consists of three buttons. The first one is used to access the page for classifying waste. The other two are generic buttons that serve to show that it is possible to add other "pages" to the application. Figure 7.6 below shows the menu interface.

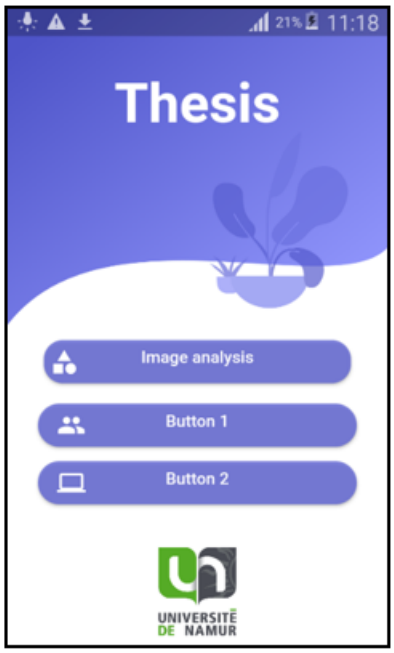

Figure 7.6: Interface menu ANN

On the other hand, in the image classification page there is a button for taking pictures at the bottom right (see Figure 7.7). When the user clicks on this button a pop-up window opens allowing him to choose whether he wants to take a picture or to pick it from his files. After selecting or taking the picture, the model estimates its category and the percentage of certainty. Figure 7.7 below shows the different steps.

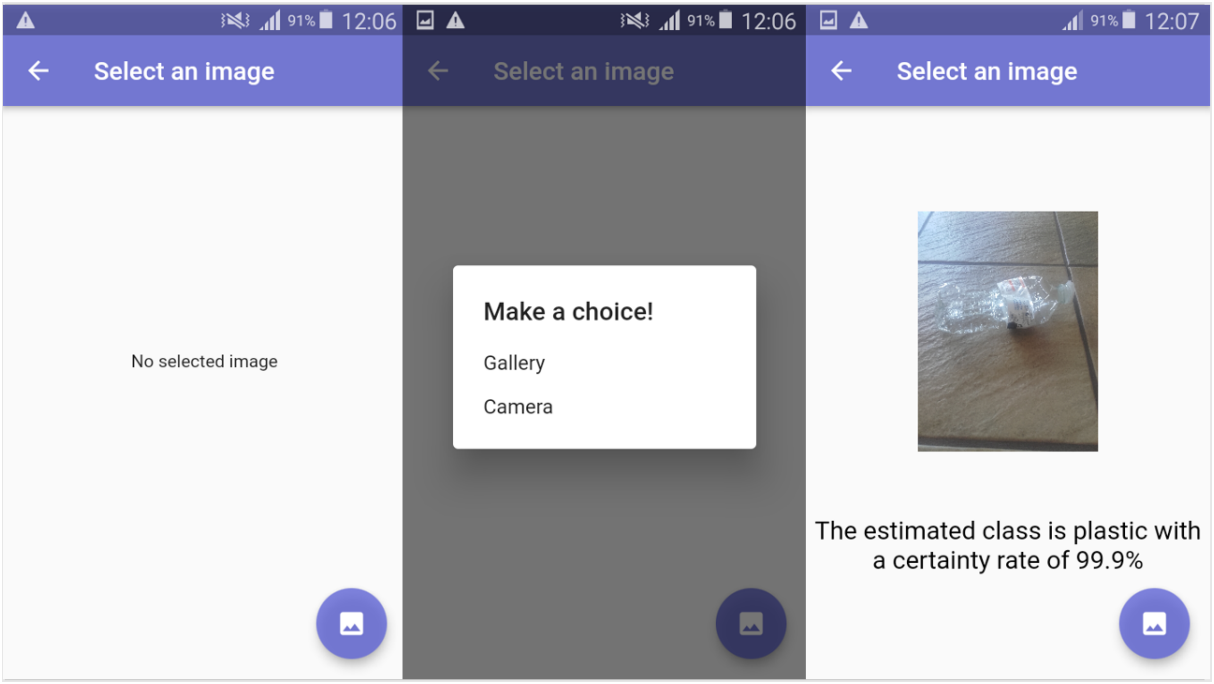

Figure 7.7: Interface image classification

7.4 Conclusion

The model and the application are obviously not perfect. Indeed, better classification results could have been achieved with more searches. However, the objective here is to provide keys to easily create one's own CNN. If the application works relatively well, the choice of a framework like Flutter limits the possibilities when creating an application. Indeed, being a higher level language, it is more difficult to modulate it precisely to expectations. Therefore, I advise to take this application as a quick way to get a concrete and functional prototype.

Chapter 8

Conclusion

The objective of this thesis is to answer the following question: ”What are the main steps needed to build and create a mobile application using deep learning with the network stored directly on the device?”. Chapter 1 introduces the problematic and explains the contributions made by this thesis.
In order to answer this question, chapter 2 of the thesis focused on the theoretical background needed to build a neural network. In this chapter, the notions of machine learning, artificial neural network, deep learning and convolutional neural network are explained. These explanations allowed to understand the mechanisms governing the creation of a model for image classification.
Then, the creation of a model for the use on a device requires to take into account the requirements of this one such as its storage limit. For this reason, Chapter 3 discussed the different mechanisms for reducing the size of the model. It introduced the notions of pruning, quantized model and architecture designed for devices.
After the explanation of these techniques, some practical aspects of the creation of a model are explained. Indeed, a big problem of model creation is the lack of data. For this reason, this chapter showed how to perform data augmentation or how to set up mechanisms to collect data. It is also in this part that the reason for storing the model on the device has been explained (compliance with RGPD).
Chapter 5 is the keystone of this thesis. Indeed, it introduces all the programming steps to create its own model. During this chapter, the creation of a model from scratch with data from a library and the creation of a model from transfer learning with data stored on a drive/hardware are explained. The set of scripts to create model from scratch can be found between appendix .3 and appendix .6 while the scripts to create a model from transfer learning can be found in the text. To complete the overview, the creation of a model from scratch with data on cloud are available in appendix .7.
Afterwards, Chapter 6 explains the creation of the application. For this part, it is more difficult to create concrete guidelines than for the creation of the model. Indeed, an application is subject to more personal choices than a model. Nevertheless, this is not pose a problem for the realization of this thesis. In fact, the aims of this chapter are on the one hand to provide a functional application by explaining it sufficiently so that it can be modified to the expectations of a person. And the other hand, It gives the main steps for the creation of one’s own application.
The chapter 7 is an application of these different chapters: an application to help with waste sorting was created. The model is created through transfer learning where several models are tested in order to find the best model. Moreover, as the training dataset is not very large, data

augmentation is applied to enlarge it and allows a better learning. The selected model is DensetNet121 with data augmentation and the SGD optimizer. This model provides an accuracy of 90.04% for the test data. The realization of this study case allows to show that the steps for the creation of the model and those for the creation of the application are functional. This case study allows to validate guidelines created during this thesis. Moreover, it shows that deep learning can help to realize good tasks for the environment. Indeed, the creation of a model with such an accuracy and with little training data shows that an application to help waste sorting is possible. Finally, the following illustration shows all the steps to create an application using CNN to sort images. This Figure 8.1 ends this thesis.

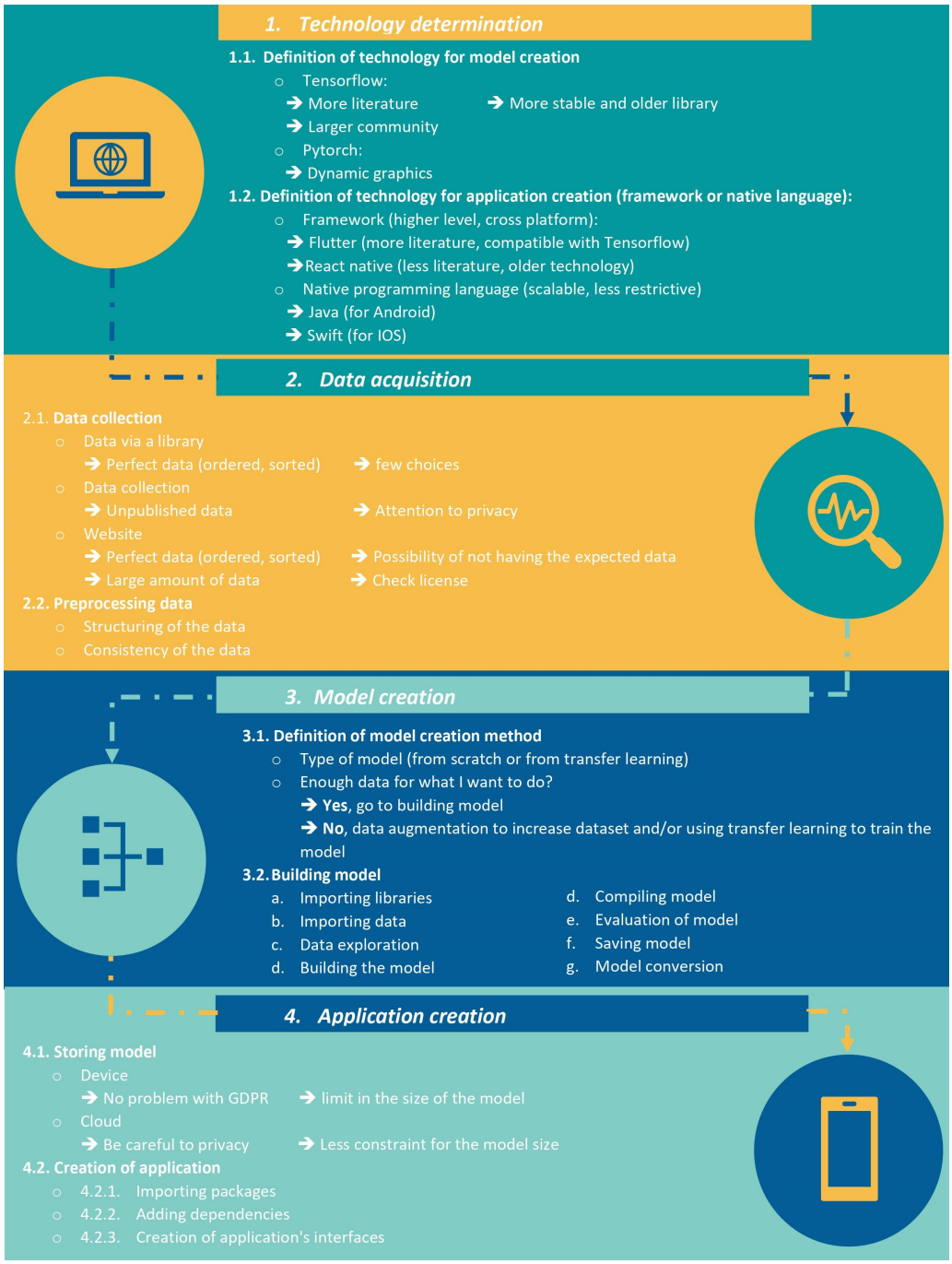

Figure 8.1: Full process

Appendices

.1 Appendix 1: Legal text for data collection

Je reconnais transmettre, volontairement, une ou des photographie(s) [XXXXXXXXXX] (ci-après le ”Contenu”) dans le cadre du travail de recherche scientifique intitulé [. . .], mené par [XX], étudiant.s au sein de la faculté [XXXXXXXXXXXXXXXXX] à l'Université de [XXXXXXXX] J'autorise, par la présente, [XX], à titre gratuit et non exclusif, à fixer, reproduire, communiquer au public et exploiter le Contenu transmis conformément au paragraphe 1er, en partie ou en totalité, sous toutes formes et sur tout support, audiovisuel ou électronique et par le biais du réseau Internet ou de tout autre réseau de transmission connu ou non connu à ce jour, à des fins de recherche scientifique.

La même autorisation est accordée, entre les mêmes parties, pour toute utilisation qui aurait des visées commerciales, de quelque nature que ce soit.

Je reconnais ne pouvoir prétendre à aucune rémunération pour la transmission du Contenu et pour la présente autorisation et je garantis que je ne suis pas lié(e) par un contrat exclusif relatif à l'utilisation de ce Contenu.

J'autorise également [XX] à concéder des licences portant sur le Contenu, similaires à la présente, mais en aucun cas plus larges, à des tiers à la présente convention.

Le Contenu que je fournis conformément au paragraphe 1er ne peut, sans que cette liste ne soit exhaustive :

- nuire à l'image de [XX], de [XXXXXXXX] ou d'un tiers ou être non conforme aux législations applicables ;
- être obscène (pornographie, pédophilie), contraire à l'ordre public et aux bonnes mœurs ;
- transmettre tout message à caractère promotionnel non sollicité ou non autorisé ;
- contenir des virus, cheval de Troie ou tout autre programme informatique dont le but est de nuire ;
- enfreindre les droits des tiers tels les droits intellectuels, le droit à la protection de la vie privée et de leurs données à caractère personnel.

Dès lors, je m'engage à ce que tout élément, information et/ou image provenant d'un tiers, inclus dans le Contenu que je transmets conformément au paragraphe 1er, est transmis avec l'autorisation préalable et expresse du tiers concerné.

Je m'engage à ce que le Contenu ne contienne aucune donnée à caractère personnel (un nom visible ou une adresse postale par exemple) me concernant ou concernant un tiers.

J'autorise également [XX] à ce que les informations (que j'ai moi-même communiquées via le formulaire) accompagnant le Contenu soient également utilisées dans le cadre de cette recherche et qu'elles puissent être publiées avec le contenu communiqué. ”

.2 Appendix 2: Script for data augmentation (data warping)

```
def data_augmentation(DATADIR, SAVINGDIR, nb_images=2,
   rotation_range=0.1, width_shift_range=0.1,
                       height_shift_range=0.1, shear_range=0.15,
   zoom_range=0.1, channel_shift_range=10,
                       horizontal_flip=True, vertical_flip =False):
  """
  Function to use data augmentation (data_warping) that you can
   fine-tune as you want
  :param DATADIR: path where the data are (string)
  :param SAVINGDIR: path where the data are stored (string)
  :param nb_images: designate the number of images to be created
   from an image
  :param rotation_range: rotation_range (int)
  :param width_shift_range: Float, 1-D array-like or int - float:
   fraction of total width, if < 1, or pixels if >= 1.(int)
  :param height_shift_range: Float, 1-D array-like or int - float:
   fraction of total height, if < 1, or pixels if >= 1.(int)
  :param shear_range: Shear Intensity (Shear angle in counter-
   clockwise direction in degrees) (float)
  :param zoom_range: Float or [lower, upper]. Range for random zoom
    (float)
  :param channel_shift_range: Range for random channel shifts. (
   float)
  :param horizontal_flip: Randomly flip inputs horizontally. (
   boolean)
  :param vertical_flip: Randomly flip inputs  vertical. (boolean)

  return no return
  NB : the structure of the file containing the images for data
   warping is
  >image_for_data_augmentation
                              > categorie1
                                        >categorie1_1.png
                                        >categorie1_2.png
                              > categorie2
                                        >categorie2_1.png
                                        >categorie2_n.png
  """
  import os
  import numpy as np
  from keras.preprocessing.image import ImageDataGenerator
  import imageio

  CATEGORIES = os.listdir(DATADIR)  # get the folders
```

```
    listdir = []  # creation list to contain categories ( one folder
     = one categorie)
    path = ''
    # modification on images
    gen = ImageDataGenerator(rotation_range=rotation_range,
     width_shift_range=0.1,
                height_shift_range=0.1, shear_range=0.15, zoom_range
     =0.1,
                channel_shift_range=10, horizontal_flip=True,
     vertical_flip =False)
    for category in CATEGORIES:
      directory = os.path.join(DATADIR, category)  # concatenation
      saving_directory = os.path.join(SAVINGDIR,
                      category)  # path to the files where the images
       will be stored with the data augmentation
      class_num = CATEGORIES.index(category)
      creation_directory = os.mkdir(saving_directory)
      nom = ''
      count = 0
      for img in os.listdir(directory):
        count += 1
        # path for each image
        image_path = os.path.join(directory, img)
        # matrix of image
        image = np.expand_dims(imageio.imread(image_path), 0)
        # get the name of the image
        part_nom = img.rsplit('.')
        # creation of new name for image with data_augmentation
        nom = part_nom[0] + '_' + str(count)
        # using data augmentation on image and save in right place
        aug_iter = gen.flow(image, save_to_dir=saving_directory,
                  save_prefix=nom)
        aug_images = [next(aug_iter)[0].astype(np.uint8) for i in
     range(nb_images)]

data_augmentation('your_data_path','path_where_your_are_stored')
```

Listing 1: Script for data warping

.3 Appendix 3: Function to Load data from libraries

```
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers
num_classes = 10
input_shape = (28, 28, 1)

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.
   load_data()
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255
# Make sure images have shape (28, 28, 1)
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
print("x_train shape:", x_train.shape)
print(x_train.shape[0], "train samples")
print(x_test.shape[0], "test samples")
```

Listing 2: Loading your data from libraries

.4 Appendix 4: Creation model from scratch

```
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers
num_classes= 10 # one for each number
# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
model_from_scratch = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)
```

Listing 3: Creation model from scratch

.5 Appendix 5: Adam's algorithm

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

Figure 2: Adam's Algoritm [21]

.6 Appendix 6: Compile and fit model from scratch for MNIST

```
#Parameters of keras example have not been changed
batch_size = 32
epochs = 15
model_from_scratch.compile(loss="categorical_crossentropy",
   optimizer="adam", metrics=["accuracy"])
model_from_scratch.fit(x_train, y_train, batch_size=batch_size,
   epochs=epochs, validation_split=0.1)
```

Listing 4: Compile and fit model from scratch for MNIST

.7 Appendix 7: Creation a model from scratch and data stored on drive/hardware data

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import os
import keras
from keras.layers import Dense, GlobalAveragePooling2D, Conv2D,
   Conv3D
from keras.applications import MobileNet, InceptionV3, ResNet50V2,
   DenseNet121, MobileNetV2, NASNetMobile
from keras.preprocessing import image
from keras.applications.mobilenet import preprocess_input
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Model
from keras.optimizers import Adam
from scipy import misc, ndimage
from keras import backend as k
% matplotlib
inline

print(tf.__version__)

train_datagen = ImageDataGenerator(preprocessing_function=
   preprocess_input)  # included in our dependencies
test_datagen = ImageDataGenerator(preprocessing_function=
   preprocess_input)

train_generator = train_datagen.flow_from_directory('
   path_to_training_data',target_size=(224, 224),color_mode='rgb',
   batch_size=32,class_mode='categorical',shuffle=True)

test_generator = test_datagen.flow_from_directory('
   path_to_test_data',target_size=(224, 224),color_mode='rgb',
   batch_size=32,class_mode='categorical',shuffle=False)

num_classes = 5  # one for each number

model_from_scratch = keras.Sequential(
    [
        keras.Input(shape=(224, 224, 3)),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
```

```
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)

model_from_scratch.compile(optimizer='SGD', loss='MSE', metrics=['
   accuracy'])  # compilation du modèle
step_size_train = train_generator.n // train_generator.batch_size
# Total number of steps (batches of samples) before declaring one
   epoch finished and starting the next epoch
number_samples_validation = test_generator.n
number_validation_steps = number_samples_validation //
   test_generator.batch_size
step_size_train = train_generator.n // train_generator.batch_size
model_from_scratch.fit(train_generator, steps_per_epoch=
   step_size_train, epochs=1, validation_data=test_generator,
                       validation_steps=number_validation_steps)
```

Listing 5: Creation a model from scratch and data stored on drive/hardware

.8 Appendix 8: File pubspec.yaml for application creation

```
name:  final_project_thesis
description: A new Flutter application.

# The following defines the version and build number for your
   application.
# A version number is three numbers separated by dots, like 1.2.43
# followed by an optional build number separated by a +.
# Both the version and the builder number may be overridden in
   flutter
# build by specifying --build-name and --build-number, respectively
   .
# In Android, build-name is used as versionName while build-number
   used as versionCode.
# Read more about Android versioning at https://developer.android.
   com/studio/publish/versioning
# In iOS, build-name is used as CFBundleShortVersionString while
   build-number used as CFBundleVersion.
# Read more about iOS versioning at
# https://developer.apple.com/library/archive/documentation/General
   /Reference/InfoPlistKeyReference/Articles/CoreFoundationKeys.
   html
version: 1.0.0+1

environment:
  sdk: ">=2.1.0 <3.0.0"

dependencies:
  flutter:
    sdk: flutter

  # The following adds the Cupertino Icons font to your application
   .
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^0.1.2
  tflite: ^1.0.4
  image_picker: ^0.6.2+4
  path_provider:
  path:

dev_dependencies:
  flutter_test:
    sdk: flutter


# For information on the generic Dart part of this file, see the
```

```
# following page: https://dart.dev/tools/pub/pubspec

# The following section is specific to Flutter.
flutter:

  # The following line ensures that the Material Icons font is
  # included with your application, so that you can use the icons
   in
  # the material Icons class.
  uses-material-design: true
  # To add assets to your application, add an assets section, like
   this:
  assets:
    - assets/

  # An image asset can refer to one or more resolution-specific "
   variants", see
  # https://flutter.dev/assets-and-images/#resolution-aware.

  # For details regarding adding assets from package dependencies,
   see
  # https://flutter.dev/assets-and-images/#from-packages

  # To add custom fonts to your application, add a fonts section
   here,
  # in this "flutter" section. Each entry in this list should have
   a
  # "family" key with the font family name, and a "fonts" key with
   a
  # list giving the asset and other descriptors for the font. For
  # example:
  # fonts:
  #   - family: Schyler
  #     fonts:
  #       - asset: fonts/Schyler-Regular.ttf
  #         style: italic
  #   - family: Trajan Pro
  #     fonts:
  #       - asset: fonts/TrajanPro.ttf
  #       - asset: fonts/TrajanPro_Bold.ttf
  #         weight: 700
  # For details regarding fonts from package dependencies,
  # see https://flutter.dev/custom-fonts/#from-packages
```

Listing 6: File pubspec.yaml for application creation

.9 Appendix 9: File Home.dart

```
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
class Home extends StatefulWidget {
  @override
  _HomeState createState() => _HomeState();
}

class _HomeState extends State<Home> {
  @override
  Widget build(BuildContext context) {
    //Variable instanciation
    var device_Size=MediaQuery.of(context).size;
    var device_height= device_Size.height;
    var device_width= device_Size.width;
    return Scaffold(
        backgroundColor: Colors.white,

        // Beginning of the Widget SingleChildSctrollView
        body:SingleChildScrollView(
          child : Center(
            //Beginning of the Containers
            child:Container(
              child: Column(

                // First part
                children: <Widget>[
                  Container(height: device_height* 1/2 /*250 */ ,
                    decoration: BoxDecoration(
                        image: DecorationImage(
                            image: AssetImage('assets/background.
   png'),
                            fit: BoxFit.fill
                        )
                    ),
                    child: Align( alignment: Alignment(0.0,-0.5),
                      child: Text("Thesis"
                        ,style: TextStyle(color: Colors.white,
   fontWeight: FontWeight.bold,fontSize: 46,),textAlign: TextAlign.
   center,),
                    ),
                ),

                // Second part
                Container(
                  width: device_height* 1/2 /*250 */,
```

```
                    padding: new EdgeInsets.all(5.0),

                    child : RaisedButton( color: Color.fromRGBO(97,
 103, 215,0.8),
                        onPressed:() {
                          Navigator.pushNamed(context,'/myapp');
                        },
                        shape:RoundedRectangleBorder(borderRadius:
BorderRadius.circular(16.0)),
                        padding: EdgeInsets.all(5.0),
                        child :Stack(children: <Widget>[
                          Align(
                              alignment: Alignment.centerLeft,
                              child: Icon(Icons.category,color:
Colors.white,)
                          ),
                          Align(
                              alignment: Alignment.center,
                              child: Text(
                                  "Image analysis",
                                  textAlign: TextAlign.center,style
 :TextStyle(color: Colors.white)
                              )
                          )
                        ],)
                    ),
                ),

                Container(
                  width: device_height* 1/2 /*250 */,
                  child : RaisedButton(color: Color.fromRGBO(97,
103, 215,0.8),
                        onPressed:() {
                          Navigator.pushNamed(context,'/Photo');
                        },
                        shape:RoundedRectangleBorder(borderRadius:
BorderRadius.circular(18.0)),
                        child :Stack(children: <Widget>[
                          Align(
                              alignment: Alignment.centerLeft,
                              child: Icon(Icons.people,color:
Colors.white,)
                          ),
                          Align(
                              alignment: Alignment.center,
                              child: Text(
                                  "Button 1",
```

```
                                    textAlign: TextAlign.center,style
 :TextStyle(color: Colors.white)
                                )
                            )
                          ],)
                      ),
                    ),

                    Container(
                      width: device_height* 1/2 /*250 */,
                      child : RaisedButton(color: Color.fromRGBO(97,
103, 215,0.8),
                          onPressed:() {
                            Navigator.pushNamed(context,'/BEP');
                          },
                          shape:RoundedRectangleBorder(borderRadius:
BorderRadius.circular(18.0)),
                          child :Stack(children: <Widget>[
                            Align(
                                alignment: Alignment.centerLeft,
                                child: Icon(Icons.computer,color:
Colors.white,)
                            ),
                            Align(
                                alignment: Alignment.center,
                                child: Text(
                                    "Button 2",
                                    textAlign: TextAlign.center,style
 :TextStyle(color: Colors.white)
                                )
                            )
                          ],)
                      ),
                    ),

                    //Last part
                    Row(
                        mainAxisAlignment: MainAxisAlignment.center,
//Center Row contents horizontally,
                        crossAxisAlignment: CrossAxisAlignment.center
, //Center Row contents vertically,
                        children: <Widget>[
                          Image.asset('assets/logo_UNamur.png',width:
 device_width*1/4,height: device_height*1/4),
                        ]
                    )

                ],
```

```
                ),
              ),
            ),
          ));
  }
}
```

Listing 7: Home.dart

.10 Appendix 10: File MyApp.dart

```
import 'package:image_picker/image_picker.dart';
import 'package:tflite/tflite.dart';
import 'dart:io';
import 'package:flutter/material.dart';
class MyApp extends StatefulWidget {
  @override
  _MyAppState createState() => _MyAppState();
}
// First part : State functions and
class _MyAppState extends State<MyApp> {
  // attribute initialization
  List _outputs;
  File _image;
  bool _loading = false;

  @override
  void initState() {
    super.initState();
    _loading = true;

    load_model().then((value) {
      setState(() {
        _loading = false;
      });
    });
  }

// Second part : creation interface function
  @override
  Widget build(BuildContext context) {
    return Scaffold(
        appBar:
      AppBar(
        title: const Text('Select an image',textAlign: TextAlign.
   start),backgroundColor: Color.fromRGBO(97, 103, 215,0.8),
      ),
```

```
      body: _loading
          ? Container(
        alignment: Alignment.center,
        child: CircularProgressIndicator(),
    )
          : Container(
        width: MediaQuery.of(context).size.width,
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.center,
          mainAxisAlignment: MainAxisAlignment.center,
          children: [

             decide_image_view(),
            SizedBox(
              height: 50,
            ),
            _outputs != null
                ?
            Text(
              output_text(),
              textAlign: TextAlign.center,
              style: TextStyle(
                color: Colors.black,
                fontSize: 20.0,
              ),


            )
                : Container()
          ],
        ),
      ),
      floatingActionButton: FloatingActionButton(backgroundColor:
   Color.fromRGBO(97, 103, 215,0.8),
        onPressed: (){
        show_choice_dialog(context);
        },
        child: Icon(Icons.image,
      ),
    ));
  }

  Future<void> show_choice_dialog(BuildContext context){
    return showDialog(context: context, builder:(BuildContext
   context) {
      return AlertDialog(
        title:Text('Make a choice!'),
        content: SingleChildScrollView(
```

```
            child: ListBody(
              children: <Widget>[
                GestureDetector(
                  child:Text('Gallery'),
                  onTap:() {
                    open_gallery(context);
                  },
                ),
                Padding(padding: EdgeInsets.all(8.0)),
                GestureDetector(
                  child:Text('Camera'),
                  onTap:(){
                    open_camera(context);
                  },
                )
              ],
            ),
          ),
        );
      });
    }


    Widget decide_image_view(){
      if(_image==null){
        return Text('No selected image');
      }
      else{
        return Image.file(_image, width: 190,height:190);
      }
    }

    String output_text(){
      String labels = "${_outputs[0]["label"]}";
      List liste= labels.split(" ");
      String label= liste[1];
      String precision =  "${_outputs[0]["confidence"].
     toStringAsFixed(3)}";
      double accuracy = double.parse(precision) *100;
      String  prec = accuracy.toString()+'%';
      String final_output = "The estimated class is " + label + "
     with a certainty rate of "+ prec;
      return final_output;
    }
  // Third part : workins functions
    pick_image() async {
      var image = await ImagePicker.pickImage(source: ImageSource.
     gallery);
```

```
    if (image == null) return null;
    setState(() {
      _loading = true;
      _image = image;
    });
    classify_image(image);
  }

  classify_image(File image) async {
    var output = await Tflite.runModelOnImage(
      path: image.path,
      numResults: 2,
      threshold: 0.5,
      imageMean: 127.5,
      imageStd: 127.5,
    );
    setState(() {
      _loading = false;
      _outputs = output;
    });
  }

  load_model() async {
    await Tflite.loadModel(
      model:  "assets/DenseNet121_SGD_data_augmentation_mai.tflite"
 ,
      labels: "assets/test_MobileNet.txt",
    );
  }

  @override
  void dispose() {
    Tflite.close();
    super.dispose();
  }

  open_gallery(BuildContext context) async{
    var image = await ImagePicker.pickImage(source :ImageSource.
   gallery);
    //pour rafraichir l'état
    if (image == null) return null;
    setState(() {
      _loading = true;
      _image = image;
    });
    classify_image(image);
    Navigator.of(context).pop(); // fermer la fenêtre de choix
  }
```

```
  open_camera(BuildContext context) async {
    var image = await ImagePicker.pickImage(source: ImageSource.
   camera);
    this.setState(() {}); //pour rafraichir l'état
    if (image == null) return null;
    setState(() {
      _loading = true;
      _image = image;
    });
    classify_image(image);
    Navigator.of(context).pop(); // fermer la fenêtre de choix
  }

}
```

Listing 8: MyApp.dart

Bibliography

[1] M Aldraimli, D Soria, J Parkinson, EL Thomas, JD Bell, MV Dwek, and TJ Chaussalet. Machine learning prediction of susceptibility to visceral fat associated diseases. Health and Technology, 10(4):925–944, 2020.

[2] Daniel Berman. What is the MIT License? Top 10 questions answered. "https://snyk.io/learn/what-is-mit-license/, 2020.

[3] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? arXiv preprint arXiv:2003.03033, 2020.

[4] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. Journal of artificial intelligence research, 16:321–357, 2002.

[5] Sébastion Collet. Qu'est-ce que le Deep Learning et comment ça marche ? "https://www.saagie.com/fr/blog/l-histoire-du-deep-learning/, 2018.

[6] IONOS company. Flutter : présentation du framework de développement d'applications multiplateforme. "https://www.ionos.fr/digitalguide/sites-internet/developpement-web/flutter-cest-quoi/, 2020.

[7] Pusher company. First steps with Flutter- Part 1: Exploring widgets. "https://pusher.com/tutorials/flutter-widgets, 2018.

[8] Li Deng and Dong Yu. Deep learning: methods and applications. Foundations and trends in signal processing, 7(3–4):197–387, 2014.

[9] Shrey Desai. What does it mean in deep learning and optimization problem that "the gradient is sparse"? "https://www.quora.com/What-does-it-mean-in-deep-learning-and-optimization-problem-that-the-gradient-is-sparse, 2019.

[10] Meenal Dhande. What is the difference between AI, machine learning and deep learning? "https://www.geospatialworld.net/blogs/difference-between-ai%EF%BB%BF-machine-learning-and-deep-learning/, 2020.

[11] Kirill Dubovikov. PyTorch vs TensorFlow — spotting the difference. "https://towardsdatascience.com/pytorch-vs-tensorflow-spotting-the-difference-25c75777377b, 2017.

[12] Benoit Frenay. Cours de machine learning, septembre 2019.

[13] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. arXiv preprint arXiv:1902.09574, 2019.

[14] Peng Gao, Qiquan Zhang, Fei Wang, Liyi Xiao, Hamido Fujita, and Yan Zhang. Learning reinforced attentional representation for end-to-end visual tracking. Information Sciences, 517:52–67, 2020.

[15] Pavel Golik, Patrick Doetsch, and Hermann Ney. Cross-entropy vs. squared error training: a theoretical and experimental comparison. In Interspeech, volume 13, pages 1756–1760, 2013.

[16] Matthijs Hollemans. Machine learning on mobile: on the device or in the cloud? "https://machinethink.net/blog/machine-learning-device-or-cloud/, 2017.

[17] Matthijs Hollemans. New mobile neural network architectures. "https://machinethink.net/blog/mobile-architectures/, 2020.

[18] Seunghoon Hong, Tackgeun You, Suha Kwak, and Bohyung Han. Online tracking by learning discriminative saliency map with convolutional neural network. In International conference on machine learning, pages 597–606. PMLR, 2015.

[19] Michal Hrabia. Deep Learning vs. Machine Learning. "https://towardsdatascience.com/deep-learning-vs-machine-learning-e0a9cb2f288, 2020.

[20] Yashwardhan Jain. Tensorflow or PyTorch : The force is strong with which one? "https://medium.com/@UdacityINDIA/tensorflow-or-pytorch-the-force-is-strong-with-which-one-68226bb7dab4, 2018.

[21] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[22] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. Supervised machine learning: A review of classification techniques. Emerging artificial intelligence applications in computer engineering, 160(1):3–24, 2007.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25:1097–1105, 2012.

[24] Namhoon Lee, Thalaiyasingam Ajanthan, Stephen Gould, and Philip HS Torr. A signal propagation perspective for pruning neural networks at initialization. arXiv preprint arXiv:1906.06307, 2019.

[25] Julie Lorenzini. Deep Learning vs. Machine Learning. "https://www.le-coin-du-digital.com/index.php/2018/07/11/deep-learning-vs-machine-learning/, 2018.

[26] Edwin Lughofer and Moamar Sayed-Mouchaweh. Adaptive and on-line learning in non-stationary environments, 2015.

[27] Marc E. McDill. cours de forest resources management syllabus, février 2021.

[28] Henri Michel. Google Colab : Le Guide Ultime. "https://ledatascientist.com/google-colab-le-guide-ultime/, 2019.

[29] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. arXiv preprint arXiv:1611.06440, 2016.

[30] Michael A Nielsen. Neural networks and deep learning, volume 25. Determination press San Francisco, CA, 2015.

[31] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. arXiv preprint arXiv:1511.08458, 2015.

[32] Uzzal Podder. How to include a custom filter in a Keras based CNN? "https://stackoverflow.com/questions/51930312/how-to-include-a-custom-filter-in-a-keras-based-cnn, 2019.

[33] Detlef Ruprecht and Heinrich Muller. Image warping with scattered data interpolation. IEEE Computer Graphics and Applications, 15(2):37–43, 1995.

[34] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 4510–4520, 2018.

[35] Ramadass Sathya and Annamma Abraham. Comparison of supervised and unsupervised learning algorithms for pattern classification. International Journal of Advanced Research in Artificial Intelligence, 2(2):34–38, 2013.

[36] Md Shahid. Convolutional Neural Network. "https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529, 2019.

[37] Lin Shao, Aishwarya Mahajan, Tobias Schreck, and Dirk J Lehmann. Interactive regression lens for exploring scatter plots. In Computer Graphics Forum, volume 36, pages 157–166. Wiley Online Library, 2017.

[38] Matthew Stewart. Simple Introduction to Convolutional Neural Networks. "https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac, 2019.

[39] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. In International conference on artificial neural networks, pages 270–279. Springer, 2018.

[40] Synopsys Editorial Team. 5 types of software licenses you need to understand. "https://www.synopsys.com/blogs/software-security/5-types-of-software-licenses-you-need-to-understand/, 2020.

[41] Muneeb ul Hassan. AlexNet – ImageNet Classification with Deep Convolutional Neural Networks. "https://neurohive.io/en/popular-networks/alexnet-imagenet-classification-with-deep-convolutional-neural-networks/, 2018.

[42] Steven Walczak and Narciso Cerpa. Artificial neural networks. In Robert A. Meyers, editor, Encyclopedia of Physical Science and Technology (Third Edition), pages 631–645. Academic Press, New York, third edition edition, 2003.

[43] Chi-Feng Wang. A Basic Introduction to Separable Convolutions. "https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728, 2018.

[44] Sebastien C Wong, Adam Gatt, Victor Stamatescu, and Mark D McDonnell. Understanding data augmentation for classification: when to warp? In 2016 international conference on digital image computing: techniques and applications (DICTA), pages 1–6. IEEE, 2016.

[45] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 4820–4828, 2016.

[46] Xin Zhang, Yongcheng Wang, Ning Zhang, Dongdong Xu, and Bo Chen. Research on scene classification method of high-resolution remote sensing images based on rfpnet. Applied Sciences, 9(10):2028, 2019.

[47] Xiaojin Zhu and Andrew B Goldberg. Introduction to semi-supervised learning. Synthesis lectures on artificial intelligence and machine learning, 3(1):1–130, 2009.

[48] Jure Zupan. Introduction to artificial neural network (ann) methods: what they are and how to use them. Acta Chimica Slovenica, 41:327–327, 1994.